CSE201 Term Project (Fall 2023)

202211062 남수민, 202211159 이호진, 202211167 장지원

Contents

A. Assignment goals

본 프로젝트는 ATM system 을 구현하는 것을 목적으로 한다. ATM system 의 전반적인 내용은 밑의 description 을 확인하면 된다.

An automated teller machine (ATM) is computerized telecommunications device that provides a financial institution's (bank) customers a method of performing financial transactions, in a public space without the need for a human bank teller. Through ATM, customers interact with a user-friendly interface that enables them to access their bank accounts and perform various transactions.

Fig 1 illustrates the scope of the ATM system. In order to use an ATM, a user (customer) must have a bank account first, and a bank issues a (debit) card that can be used to access the bank account through ATMs. Note that ATMs do not maintain any user information locally such as account numbers or passwords. For this reason, ATMs are connected to external banks to retrieve and verify necessary user information to perform transaction requests.

A user may access several different types of ATMs. Each ATM is managed by a primary bank, and some ATMs may allow users to perform transactions with other non-primary bank accounts as well (e.g., withdraw funds from Daegu Bank account using KookMin Bank ATM). An ATM keeps a certain amount of available cash that can be immediately provided to users. The available cash inside ATMs may increase or decrease as multiple users perform a series of transactions such as deposit or withdraw cash. An ATM also allows non-cash transactions such as bank transfers or check deposits.

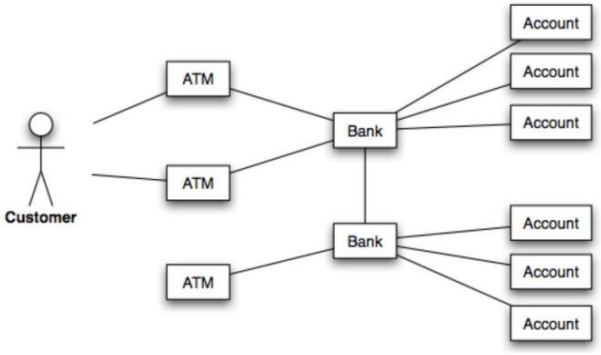


Figure 1 UML Use Case Diagram

B. Requirements (REQ)

ATM system 을 위한 요구사항은 다음과 같다.

1. System Setup

(REQ1.1) An ATM has a 6-digit serial number that can be uniquely identified among all ATMs (e.g., 315785).
(REQ1.2) An ATM is set to one of the following types: (1) Single Bank ATM, (2) Multi-Bank ATM.
(REQ1.3) An ATM may support either unilingual or bilingual languages.
(REQ1.4) A Bank deposits a certain amount of cash to an ATM to serve users.
(REQ1.5) A Bank can open an Account for user with necessary information to perform bank services.
(REQ1.6) A user may have multiple Accounts in a Bank
(REQ1.7) A user may have Accounts in multiple Banks
(REQ1.8) Each ATM have several types of transaction fees as follows:
(REQ1.9) An admin can access the menu of "Transaction History" via an admin card (See REQ Display of Transaction History)
(REQ1.10) An ATM only accepts and returns the following types of cashes

2. ATM Session

(REQ2.1) A session starts when a user inserts a card.
(REQ2.2) A session ends whenever a user wishes (e.g., by choosing a cancel button) or there are some exceptional conditions detected by the ATM (e.g., no cash available).
(REQ2.3) When a session ends, the summary of all transactions performed in a session must be displayed
(REQ2.4) Each transaction has a unique identifier across all sessions.

3. User Authorization

(REQ3.1) An ATM checks if the inserted card is valid for the current type of ATM. - See REQ in System Setup which card is considered valid
(REQ3.2) If an invalid card is inserted, the ATM shall display an appropriate error message (e.g., Invalid **Card).**
(REQ3.3) An ATM shall ask a user to enter the password (e.g., Enter Password), and verify if the password is correct
(REQ3.4) If the entered password is incorrect, the ATM shall display an appropriate error message (e.g., Wrong Password).
(REQ3.5) If a user enters wrong passwords 3 times in a row, a session is aborted, and return the card to the user.

4. Deposit

(REQ4.1) An ATM shall take either cash or check from a user.
(REQ4.2) An ATM shall display an appropriate error message if the number of the inserted cash or checks exceeds the limit allowed by the ATM.
(REQ4.3) Once cash or checks are accepted by ATM, the transaction must be reflected to the bank account as well (i.e., the same amount of fund must be added to the corresponding bank account).
(REQ4.4) Some deposit fee may be charged (See REQ in System Setup)
(REQ4.5) The deposited cash increase available cash in ATM that can be used by other users.
(REQ4.6) The deposited check does not increase available cash in ATM that can be used by other users.

5. Withdrawal

(REQ5.1) An ATM shall ask a user to enter the amount of fund to withdraw.
(REQ5.2) An ATM shall display an appropriate error message if there is insufficient fund in the account or insufficient cash in the ATM.
(REQ5.3) Once the withdrawal is successful, the transaction must be reflected in the bank account as well (i.e., the same amount of fund must be deducted from the corresponding bank account).
(REQ5.4) Some withdrawal fee may be charged (See REQ in System Setup).
(REQ5.5) The cash withdrawal lower available cash in the ATM that can be used by other users.

6. Transfer

(REQ6.1) An ATM shall ask a user to choose the types of transfer either cash transfer or account fund transfer.
(REQ6.2) For both cash and account transfers, an ATM shall ask the destination account number where the fund is to be transferred.
(REQ6.3) For cash transfer, an ATM shall ask the user to insert the cash including the transaction fees and verify if the amount of the inserted cash is correct. All inserted cash excluding the transaction fee shall be transferred.
(REQ6.4) For account transfer, an ATM shall ask the source account number, and the amount of fund to be transferred.
(REQ6.5) Some transfer fee may be charged (See REQ in System Setup).
(REQ6.6) The inserted cash for transfer increase available cash in ATM that can be used by other users.
(REQ6.7) Once the transfer is successful, the transaction must be reflected to the bank account as well (i.e., the same amount of fund must be deducted from the source bank account, and then added to the destination bank account).

7. Display of Transaction History (Admin Menu)

(REQ7.1) When a session is started by an admin by inserting an admin card (See REQ in System Setup), an ATM displays a menu of "Transaction History" only.
(REQ7.2) When the "Transaction History" menu is selected, an ATM display the information of all transactions from all users from the beginning of the system start
(REQ7.3) The "Transaction History" information shall be outputted to the external file (e.g., txt file).

8. Multi-language support

(REQ8.1) An ATM that is configured with the bilingual support shall provide an option for a user to choose the preferred language either English or Korean.
(REQ8.2) Once a certain language is chosen, all menus must be displayed using the chosen language

9. Exception Handling

(REQ9.1) An ATM shall display an appropriate message for each exception scenario (both explicitly stated in this document and implicitly assumed ones), and take an appropriate action (e.g., end a session).

10. Display of Account/ATM Snapshot

(REQ10.1) When a particular character (e.g. 'x') is given as a console input during the program execution, the following information shall be displayed to the console.

C. Instruction of Final version source code

다음은 Instruction 에 대한 설명이다.

1. header file, class declaration

먼저 standard iostream, string, fstream, vector header file 들을 include 해주었다. 다음으로 우리가 구현한 class (Account, Bank, ATM)들과 txt 파일 사용을 위한 ofstream Summary class 를 선언해주었다.

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>


using namespace std;

class Account;
class Bank;
class ATM;
ofstream Summary;
```

2. Account, Bank, ATM class definition

다음으로 Account, Bank, ATM class의 definition을 작성하였다. 먼저 Account class의 definition이다.

private:

string name: Account 주 이름이다.

string account_num: Account 번호이다. 12자리로 구성되어 있다.

int password: Account의 비밀 번호이다. 4자리로 구성되어 있다.

int balance: Account에 남아있는 balance이다. 이는 출금, 입금, 계좌 이체 과정에서 계속 update 된다. *string bank*: Account의 bank이다.

public:

Account(string bank_name, string user_name, string account_number, int user_password, int available_funds): Account class의 construct이다. bank_name, user_name, account_number, user_password, available_funds를 argument로 가진다.

string getBankName(): Account의 Bank 이름을 반환하는 함수이다.

string getAccountNum(): Account의 번호를 반환하는 함수이다.

int getPassword(): Account의 비밀번호를 반환하는 함수이다.

int getBalance(): Account의 잔액을 반환하는 함수이다.

void setBalance(int update_balance): Account의 잔액을 setting하는 함수이다. update_balance를

argument로 가진다. 이 argument를 바탕으로 잔액을 setting한다.

string getName(): Account의 주 이름을 반환하는 함수이다.

```
class Account {
private:
    string name;        // 계좌 주 이름
    string account_num; // 계좌번호
    int password;       // 비밀번호
    int balance;        // 잔액
    string bank;        // 계좌 은행

public:
    Account(string bank_name, string user_name, string account_number,
        int user_password, int available_funds);
    string getBankName();
    string getAccountNum();
    int getPassword();
    int getBalance();
    void setBalance(int update_balance);
    string getName();
};
```

다음은 Bank class의 definition이다.

private:

string bank_name: 해당 은행의 이름이다.

public:

vector<Account> Account_list*: 해당 bank의 account list이다.

static vector<ATM> ATM_list*: 모든 ATM의 list이다. static으로 저장되어 있다.

static vector<Bank> Bank_list*: 모든 bank의 list이다. static으로 저장되어 있다.

Bank(const string): Bank class의 constructor이다. name을 argument로 갖는다.

*void addATM(ATM *atm)*: ATM_list에 ATM을 하나씩 추가해주는 역할을 하는 함수이다.

*void setAccount(Account *)*: Bank에 account를 설정해주는 함수이다.

void getAccount(): Bank에 있는 모든 account를 반환해주는 함수이다.

*void setDeposit(Account *, int money)*: deposit 결과를 balance에 반영해주는 함수이다. money argument를 통해서 balance를 조정한다. balance가 bank를 거쳐서 조정되도록 구현하기 위해서 이 함수를 작성하였다.

*void setWithdrawal(Account *account, int money)*: withdrawal 결과를 balance에 반영해주는 함수이다. money argument를 통해서 balance를 조정한다. balance가 bank를 거쳐서 조정되도록 구현하기

위해서 이 함수를 작성하였다.

*void setTransfer(Account *account, int money)*: transfer 결과를 balance에 반영해주는 함수이다. money argument를 통해서 balance를 조정한다. balance가 bank를 거쳐서 조정되도록 구현하기 위해서 이 함수를 작성하였다.

*bool verifyPassword(Account *account, int pw)*: 비밀번호를 확인하는 함수이다. 비밀번호가 맞다면 True를 틀리다면 False를 반환한다.

*Account *findAccount(string ID)*: string ID argument와 일치하는 계좌를 반환해주는 함수이다. 입력받은 계좌의 porinter를 구하기 위해서 이 함수를 작성하였다.

string getBankName(): bank 이름을 반환하는 함수이다.

void printAllAccounts(): 해당 bank가 주 거래 은행인 모든 account를 반환하는 함수이다.

```
class Bank {
private:
    string bank_name;
public:
    vector<Account*> Account_list;
    static vector<ATM*> ATM_list; // 모든 ATM 리스트
    static vector<Bank*> Bank_list; // 모든 bank 의 list -> atm 에서 사용하기
위함
    Bank(const string);
    void addATM(ATM* atm);      // atm_list 에 ATM 을 추가하는 함수
    void setAccount(Account*); // 계좌 추가
    void getAccount();          // 관리하는 계좌들 출력
    void setDeposit(Account*, int money); // deposit 결과 계좌에 반영
    void setWithdrawal(Account* account, int money); // withdrawal 결과 계좌에
반영
    void setTransfer(Account* account, int money); // transfer 결과 계좌에 반영
    bool verifyPassword(Account* account, int pw); // pw 가 account 비번 맞는지
확인
    Account* findAccount(string ID);
    string getBankName(); // bank name return
    // vector<Bank*> getAllBank();
    void printAllAccounts();
};
```

마지막으로 ATM class의 definition이다.

private:

int serial_type: 해당 은행의 이름이다.

Bank serial_primary_bank*: ATM의 primaty bank의 pointer타입이다.

int count = 0: count를 위한 변수이다.

int num_of_1000: ATM에 있는 1000원권의 개수이다.

int num_of_5000: ATM에 있는 5000원권의 개수이다.

int num_of_10000: ATM에 있는 10000원권의 개수이다.

int num_of_50000: ATM에 있는 50000원권의 개수이다.

*string** SessionSummary = new string * [20]*: ATM의 transaction histories를 저장하기 위한 리스트이다. *int trans = 0*: ATM에서 일어난 transaction의 수를 counting하기 위한 변수이다.

int withdrawal_count: ATM에서 일어난 withdrawal수를 counting하기 위한 변수이다.

public:

int serial_num: argument로 받은 ATM의 serial number를 저장하기 위한 변수이다.

Bank current_bank*: argument로 받은 ATM의 primary bank pointer를 저장하기 위한 변수이다.

Account current_acc*: ATM에 입력된 account의 bank pointer를 저장하기 위한 변수이다.

vector<Bank> Bank_list*: ATM constructor에서 argument로 받은 모든 bank의 list를 저장하기 위한 vector이다.

int available_cash: ATM에 남아있는 금액이다.

ATM(Bank primary_bank, int serial_number, int type, int language, int initial_1000, int initial_5000, int initial_10000, int initial_50000, vector<Bank*> allBanks)* : ATM constructor이다. ATM의 primary bank poiner, ATM의 serial number, ATM의 type (single bank인지 multi bank인지), ATM의 language type(unilingual을 지원하는지, bilingual을 지원하는지), ATM에 남아있는 각 지폐의 개수, 모든 bank의 list를 argument로 갖는다.

void addBank(): Bank_list에 bank를 하나씩 추가해주는 역할을 하는 함수이다.

bool authorization(string account_num): 인증을 진행하는 함수이다. 올바른 카드가 입력되었는지, 올바른 비밀번호가 입력되었는지 확인하는 과정을 진행한다.

void deposit(): 예금 과정을 진행하는 함수이다.

void withdrawal(): 출금 과정을 진행하는 함수이다.

bool withdrawal_fee(): 출금 과정에서 발생하는 수수료 처리를 진행하는 함수이다.

void transfer(): 계좌이체 과정을 진행하는 함수이다.

void StartSession(): ATM의 모든 과정을 관리하는 함수이다. 이 함수를 통해 *void deposit(), void withdrawal(), void transfer()*에 접근하여 각각의 과정을 처리한다.

bool authorization(): 인증을 진행하는 함수이다. 올바른 카드가 입력되었는지, 올바른 비밀번호가 입력되었는지 확인하는 과정을 진행한다.

void admin(): 카드 입력 부분에서 admin의 정보가 들어오면 해당 ATM의 모든 history를 출력해주어야 한다. 이를 처리하기 위한 함수이다.

void ATMHistory(string trans, Account account, Account* desinationaccount, int amount)*: ATM의 모든 history를 관리하는 함수이다. trans(ATM에서 일어난 transaction의 수), account(계좌), desinationaccount(transfer 과정에서의 입금 계좌), amount(거래 금액)를 argument로 갖는다.

void PrintHistory(int session): ATM의 모든 history를 출력하는 함수이다.

int serial_lang_type: language type을 저장하기 위한 변수이다. unilingual이면 1, bilingual이면 2가 저장된다.

int language_you_want: bilingual일 때 추가 선택(영어, 한국어)을 저장하기 위한 변수이다. 영어 보기를 선택했다면 2, 한국어 보기를 선택했다면 1이 저장된다.

void setLanguage(): *int language_you_want*변수를 setting하기 위한 함수이다.

void DisplayOfAccountATMEng(): input에 특정 문자가 입력되면 존재하는 ATM과 account의 모든 정보가 출력되어야 한다. 이를 위한 함수이다. 사용자가 영어 보기를 선택했을 때 동작한다.

void DisplayOfAccountATMKo(): input에 특정 문자가 입력되면 존재하는 ATM과 account의 모든 정보가 출력되어야 한다. 이를 위한 함수이다. 사용자가 한국어 보기를 선택했을 때 동작한다.

```
class ATM {
private:
  int serial_type;           // single bank 이면 1, multi bank 이면 2
  Bank *serial_primary_bank; // primary bank 의 ptr
  int count = 0;
  int num_of_1000;
  int num_of_5000;
  int num_of_10000;
  int num_of_50000;
  string **SessionSummary = new string *[200];
  int trans = 0;
  int withdrawal_count = 0;

public:
  int serial_num;       // serial number
  Bank *current_bank;   // 현재 사용중인 bank
  Account *current_acc; // 현재 사용중인 account
  vector<Bank *> Bank_list;
  int available_cash; // ATM 에 남은 돈
  ATM(Bank *primary_bank, int serial_number, int type, int language,
      int initial_1000, int initial_5000, int initial_10000, int initial_50000,
      vector<Bank *> allBanks); // multi bank ATM constructor
  void addBank();
  bool authorization(string account_num);
  void deposit(); // 여기까지 구현함!!
  void withdrawal();
  bool withdrawal_fee();
  void transfer();
  void StartSession();
  bool authorization();
  void admin();
  void ATMHistory(string trans, Account *account, Account *desinationaccount,
                  int amount, string type);
  void PrintHistory(int session);
  int serial_lang_type; // unilingual  이면 1, bilingual  이면 2
```

```
  int language_you_want;
  void setLanguage();
  void DisplayOfAccountATMEng(); // 이걸 cin 별로 다 만들어 놓기
  void DisplayOfAccountATMKo();
};
```

3. Account, Bank, ATM class implementation

다음은 Account, Bank, ATM 의 Implementation 이다. 위 Definition 을 바탕으로 구현해보았다. 먼저 Account 의 Implementaion 이다.

```
Account::Account(string bank_name, string user_name, string account_number,
                 int user_password, int available_funds) {
  this->bank = bank_name;
  this->name = user_name;
  this->account_num = account_number;
  this->password = user_password;
  this->balance = available_funds;
  cout << "[Account Constructor] Account ID: " << this->account_num << endl;
}

string Account::getBankName() { return this->bank; }
string Account::getAccountNum() { return this->account_num; }
int Account::getPassword() { return this->password; }
int Account::getBalance() { return this->balance; }
void Account::setBalance(int update_balance) { this->balance = update_balance; }
string Account::getName() { return this->name; }
```

다음은 Bank 의 Implementation 이다.

```
vector<Bank *> Bank::Bank_list;
vector<ATM *> Bank::ATM_list;

Bank::Bank(const string name) {
  this->bank_name = name;
  Bank_list.push_back(this); // bank_list stack 에 생성
  cout << "[Bank Constructor] Bank name: " << this->bank_name << endl;
}
void Bank::addATM(ATM *atm) { // atm 추가, atm constructor 에서 알아서 돌아감
  ATM_list.push_back(atm);
}
void Bank::setAccount(Account *account) { Account_list.push_back(account); }

// vector<Bank*> Bank::getAllBank() { return Bank_list; }

void Bank::getAccount() { return; }
```

```
void Bank::setDeposit(Account *account, int money) {
  account->setBalance(account->getBalance() + money);
}

void Bank::setWithdrawal(Account *account, int money) {
  account->setBalance(account->getBalance() - money);
}

void Bank::setTransfer(Account *account, int money) {
  account->setBalance(account->getBalance() + money);
}

bool Bank::verifyPassword(Account *account, int pw) {
  if (account->getPassword() == pw) {
    return true;
  }
  return false;
}

Account *Bank::findAccount(string ID) {
  for (int i = 0; i < Account_list.size(); i++) {
    if (Account_list[i]->getAccountNum() == ID) {
      return Account_list[i];
    }
  }
  return nullptr;
}

string Bank::getBankName() { return this->bank_name; }

void Bank::printAllAccounts() {
  cout << "num of accounts: " << Account_list.size() << endl;
  for (int i = 0; i < Account_list.size(); i++) {
    cout << (i + 1) << ". account number: " << Account_list[i]->getAccountNum()
         << endl;
  }
}
```

마지막으로 ATM 의 Implementation 이다.

```
ATM::ATM(Bank *primary_bank, int serial_number, int type, int language,
         int initial_1000, int initial_5000, int initial_10000,
         int initial_50000, vector<Bank *> allBanks) {
  // primary bank setting
  serial_primary_bank = primary_bank; // atm 의 bank
  serial_primary_bank->addATM(
      this); // Bank class 의 static ATM_list 에 해당 ATM 추가
```

```
  //  serial number setting
  serial_num = serial_number;
  // bank type setting
  serial_type = type;
  serial_lang_type = language;
  // initial cash setting
  this->num_of_1000 = initial_1000;   // 지폐 갯수
  this->num_of_5000 = initial_5000;   // 지폐 갯수
  this->num_of_10000 = initial_10000; // 지폐 갯수
  this->num_of_50000 = initial_50000; // 지폐 갯수
  // bank list
  this->Bank_list = allBanks;
  // session summary 저장해놓을 곳 initialize
  cout << "[ATM Constructor] ATM serial number: " << this->serial_num << endl;
  cout << "serial type: " << serial_type << endl;
  cout << "serial lang: " << serial_lang_type << endl;
}

void ATM::setLanguage() {
  if (this->serial_lang_type == 2) {
    cout
        << "Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?"
        << endl;
    cout << "1.Korean/한국어\t2.English/영어";
    cin >> this->language_you_want;
  } else {
    this->language_you_want = 2;
  }
}

void ATM::ATMHistory(string tran, Account *account, Account *destinationaccount,
                     int amount, string type) {
  SessionSummary[trans] = new string[9];
  SessionSummary[trans][0] = to_string(trans);
  SessionSummary[trans][1] = tran;
  SessionSummary[trans][8] = type;
  if (account != nullptr) {
    SessionSummary[trans][2] = account->getBankName();
    SessionSummary[trans][3] = account->getAccountNum();
  } else {
    SessionSummary[trans][2] = "--";
    SessionSummary[trans][3] = "--";
  }
  if (destinationaccount != nullptr) {
    SessionSummary[trans][4] = destinationaccount->getBankName();
    SessionSummary[trans][5] = destinationaccount->getAccountNum();
  } else {
    SessionSummary[trans][4] = "--";
```

```
    SessionSummary[trans][5] = "--";
  }
  SessionSummary[trans][6] = to_string(amount);
  SessionSummary[trans][7] = to_string(this->available_cash);

  trans++;
}

// StartSession
// void ATM::StartSession(Bank** bank_list) { // bank list를 파라미터로 받아옴
// StartSession
// void ATM::StartSession(Bank** bank_list) { // bank list를 파라미터로 받아옴
void ATM::StartSession() {
  if (this->language_you_want == 2) {
    int check_acc = 0;
    string account_num; // 계좌번호
    // string bank_name; // 이 계좌의 은행이 어디인지 확인하는 변수
    cout << "Insert card: ";
    cin >> account_num; // 계좌번호 입력받음
    if (account_num == "-1") {
      DisplayOfAccountATMEng();
      cout << "Insert card: ";
      cin >> account_num; // 계좌번호 입력받음
    }
    if (account_num == "000000") {
      admin();
      return;
    }

    // 이케함, len 따로 구해도 됨
    for (int i = 0; i < Bank_list.size(); i++) {
      this->current_acc = Bank_list[i]->findAccount(account_num);
      if (current_acc != nullptr) { // 찾으면 break
        current_bank = Bank_list[i];
        check_acc = 1;
        break;
      }
    }

    if (check_acc == 0) {
      cout << "[Error] Such account does not exist." << endl;
      StartSession();
      return;
    }
    if (this->authorization() == false) {
      cout << "Authorization failed" << endl;
      cout << "Session finished" << endl;
      return;
```

```
    }
    int action = 0;
    int session = trans;
    while (action != 4) {
      cout << endl;
      cout << "================================" << endl;
      cout << "Choose transaction" << endl;
      cout << endl;
      cout << "1.Deposit\t 2.Withdrawal\t3.Transfer\t4.Exit : ";
      cin >> action;
      if (action == -1) {
        DisplayOfAccountATMEng();
        cout << "Choose transaction" << endl;
        cout << "1.Deposit\t 2.Withdrawal\t3.Transfer\t4.Exit : ";
        cin >> action;
      }
      cout << endl;
      if (action == 1) {
        deposit();
      } else if (action == 2) {
        withdrawal();
      } else if (action == 3) {
        transfer();
      } else if (action == 4) {
        PrintHistory(session);
        break;
      } else {
        cout << "[ERROR] Please enter 1~4" << endl;
      }
    }
    cout << "Session finished" << endl;
    return;
  }
  if (this->language_you_want == 1) {
    // bank list에 item 어떻게 담아 넣을지 모르겠음
    int check_acc = 0;
    string account_num; // 계좌번호
    // string bank_name; // 이 계좌의 은행이 어디인지 확인하는 변수
    cout << "카드를 입력하시오 : ";
    cin >> account_num; // 계좌번호 입력받음
    if (account_num == "-1") {
      DisplayOfAccountATMKo();
      cout << "카드를 입력하시오 : ";
      cin >> account_num; // 계좌번호 입력받음
    }
    if (account_num == "000000") {
      admin();
      return;
```

```
  }
  for (int i = 0; i < Bank_list.size(); i++) {
    this->current_acc = Bank_list[i]->findAccount(account_num);
    if (current_acc != nullptr) { // 찾으면 break
      current_bank = Bank_list[i];
      check_acc = 1;
      break;
    }
  }

  if (check_acc == 0) {
    cout << "잘못된 계좌번호입니다." << endl;
    StartSession();
    return;
  }
  if (this->authorization() == false) {
    cout << "인증 오류" << endl;
    cout << "세션이 종료됩니다." << endl;
    return;
  }
  int action = 0;
  int session = trans;
  while (true) {
    cout << "진행하실 거래를 선택해주세요." << endl;
    cout << "1.예금\t 2.출금 \t3.이체 \t4.취소 : ";
    cin >> action;
    if (action == -1) {
      DisplayOfAccountATMKo();
      cout << "진행하실 거래를 선택해주세요." << endl;
      cout << "1.예금\t 2.출금 \t3.이체 \t4.취소 : ";
      cin >> action;
    }
    cout << endl;
    if (action == 1) {
      deposit();
    } else if (action == 2) {
      withdrawal();
    } else if (action == 3) {
      transfer();
    } else if (action == 4) {
      PrintHistory(session);
      break;
    } else {
      cout << "[오류] 거래 1~4 중에 선택하시오." << endl;
    }
  }
  cout << "세션이 종료됩니다." << endl;
  return;
```

```
  }
}
// stop

// 3. User Authorization
bool ATM::authorization() {
  if (this->language_you_want == 2) {
    cout << "Authorization Started" << endl;
    int password;
    // REQ3.1 카드가 atm 기기의 타입에 맞는 유효한 카드인지 확인함
    // REQ3.2 유효하지 않으면 -> display error message(ex.”Invalid Card”)
    if ((this->serial_type == 1) &&
        (current_acc->getBankName() != serial_primary_bank->getBankName())) {
      // ATM이 single bank ATM이면서 입력받은 계좌의 은행이 primary bank가 아닐
      // 때
      cout << "[ERROR] Invalid Card: the ATM is a single bank atm, and the "
              "card's bank is not the primary bank"
           << endl;
      return false;
    }
    // REQ3.3 사용자가 비밀번호를 입력하게하여 맞는지 확인 ( bank로 정보 넘겨서
    // 확인받음 ) R EQ3.4 비밀번호 틀렸으면 -> display error message
    cout << "Insert password: ";
    cin >> password;
    if (password == -1) {
      DisplayOfAccountATMEng();
      cout << "Insert password: ";
      cin >> password;
    }
    cout << endl;
    int pw_count = 0;
    while ((current_bank->verifyPassword(current_acc, password)) != true) {
      cout << "[ERROR] Incorrect Password" << endl;
      pw_count++;
      // REQ3.5 연속으로 3번 틀리면 session 끝나야함
      if (pw_count == 3) {
        cout << "Incorrect Password 3 times in a row..." << endl;
        return false; // 여기선 비밀번호 틀림... -> end session
      }
      // 비밀번호 입력 횟수 3번 미만이면 다시 비밀번호 입력받음
      cout << "Insert password: ";
      cin >> password;
      if (password == -1) {
        DisplayOfAccountATMEng();
        cout << "Insert password: ";
        cin >> password;
      }
      cout << endl;
```

```
    }
    return true; // 여기선 비밀번호 맞음...
  } else if (this->language_you_want == 1) {
    cout << "인증 절차를 시작합니다." << endl;
    int password;
    // REQ3.1 카드가 atm 기기의 타입에 맞는 유효한 카드인지 확인함
    // REQ3.2 유효하지 않으면 -> display error message(ex.”Invalid Card”)
    if ((this->serial_type == 1) &&
        (current_acc->getBankName() != serial_primary_bank->getBankName())) {
      // ATM 이 single bank ATM 이면서 입력받은 계좌의 은행이 primary bank 가 아닐
      // 때
      cout << "[오류] 유효하지 않은 카드입니다: 이 ATM 은 단일 거래만 "
              "가능하지만, 카드의 은행은 주 거래 은행이 아닙니다."
           << endl;
      return false;
    }
    // REQ3.3 사용자가 비밀번호를 입력하게하여 맞는지 확인 ( bank 로 정보 넘겨서
    // 확인받음 ) R EQ3.4 비밀번호 틀렸으면 -> display error message
    cout << "비밀번호를 입력해주세요." << endl;
    cin >> password;
    if (password == -1) {
      DisplayOfAccountATMKo();
      cout << "비밀번호를 입력해주세요." << endl;
      cin >> password;
    }
    cout << endl;
    int pw_count = 0;
    while ((current_bank->verifyPassword(current_acc, password)) != true) {
      cout << "[오류] 비밀번호가 틀렸습니다." << endl;
      pw_count++;
      // REQ3.5 연속으로 3 번 틀리면 session 끝나야함
      if (pw_count == 3) {
        cout << "3 회 연속으로 비밀번호가 틀려 세션이 종료됩니다." << endl;
        return false; // 여기선 비밀번호 틀림... -> end session
      }
      // 비밀번호 입력 횟수 3 번 미만이면 다시 비밀번호 입력받음
      cout << "비밀번호를 입력해주세요. " << endl;
      cin >> password;
      if (password == -1) {
        DisplayOfAccountATMKo();
        cout << "비밀번호를 입력해주세요." << endl;
        cin >> password;
      }
      cout << endl;
    }
    return true; // 여기선 비밀번호 맞음...
  } else {
    return false;
```

```
  }
}

// 4. Deposit
void ATM::deposit() {
  int deposit_type;      // 1 for cash deposit, 2 for check deposit
  int deposit_amount;    // deposit 장수
  int total_deposit = 0; // deposit 총액
  int cash_limit = 30;
  int check_limit = 50;
  int deposit_1000;
  int deposit_5000;
  int deposit_10000;
  int deposit_50000;
  if (this->language_you_want == 2) {
    // REQ4.1 현금/수표 입력받음
    cout << "What are you depositing?";
    cout << "1.cash deposit\t2.check deposit\t3. Exit" << endl;
    cin >> deposit_type;
    if (deposit_type == -1) {
      DisplayOfAccountATMEng();
      cout << "What are you depositing?";
      cout << "1.cash deposit\t2.check deposit\t3. Exit" << endl;
      cin >> deposit_type;
    }
    cout << endl;

    // cash deposit 선택함
    if (deposit_type == 1) {
      type = "Cash Deposit";
      cout << "How much do you want to deposit?" << endl;
      cout << "Insert the amount of 1000won cash : ";
      cin >> deposit_1000;
      if (deposit_1000 == -1) {
        DisplayOfAccountATMEng();
        cout << "How much do you want to deposit?" << endl;
        cout << "Insert the amount of 1000won cash : ";
        cin >> deposit_1000;
      }
      cout << endl;
      if (deposit_1000 < -1) { // 0 이하의 정수 받았을 때
        cout << "Error : Amount can't be negative." << endl;
        cout << endl;
        return;
      }
      cout << "Insert the amount of 5000won cash : ";
      cin >> deposit_5000;
      if (deposit_5000 == -1) {
```

```
      DisplayOfAccountATMEng();
      cout << "How much do you want to deposit?" << endl;
      cout << "Insert the amount of 5000won cash : ";
      cin >> deposit_5000;
    }
    cout << endl;
    if (deposit_5000 < -1) { // 0 이하의 정수 받았을 때
      cout << "Error : Amount can't be negative." << endl;
      cout << endl;
      return;
    }
    cout << "Insert the amount of 10000won cash : ";
    cin >> deposit_10000;
    if (deposit_10000 == -1) {
      DisplayOfAccountATMEng();
      cout << "How much do you want to deposit?" << endl;
      cout << "Insert the amount of 10000won cash : ";
      cin >> deposit_10000;
    }
    cout << endl;
    if (deposit_10000 < -1) { // 0 이하의 정수 받았을 때
      cout << "Error : Amount can't be negative." << endl;
      cout << endl;
      return;
    }
    cout << "Insert the amount of 50000won cash : ";
    cin >> deposit_50000;
    if (deposit_50000 == -1) {
      DisplayOfAccountATMEng();
      cout << "How much do you want to deposit?" << endl;
      cout << "Insert the amount of 50000won cash : ";
      cin >> deposit_50000;
    }
    cout << endl;
    if (deposit_50000 < -1) { // 0 이하의 정수 받았을 때
      cout << "Error : Amount can't be negative." << endl;
      cout << endl;
      return;
    }

    deposit_amount =
        deposit_1000 + deposit_5000 + deposit_10000 + deposit_50000;
    total_deposit = deposit_1000 * 1000 + deposit_5000 * 5000 +
                    deposit_10000 * 10000 + deposit_50000 * 50000;
    // Exceptional Handling : 이체량이 0 원이면 Error Message
    if (deposit_amount == 0) {
      cout << "[ERROR] You should deposit least 1000 for transaction."
           << endl;
```

```
      cout << endl;
      return;
    }
    // REQ4.2 개수 제한 넘으면 display error message
    if (deposit_amount > cash_limit) {
      cout << "[ERROR] Your deposit amount is over 30(cash deposit limit)."
           << endl;
      return;
    }

    // REQ4.4 이체 수수료 부과될 수 있음 ( non - primary bank deposit 일 때)
    if ((this->serial_type == 2) &&
        (current_acc->getBankName() != serial_primary_bank->getBankName())) {
      int confirm_fee = 2;
      cout << "You are charged for deposit fee of 1000won." << endl;
      cout << "1.Insert addtional cash \t 2.Exit \t : ";
      cin >> confirm_fee;
      if (confirm_fee == -1) {
        DisplayOfAccountATMEng();
        cout << "You are charged for deposit fee of 1000won." << endl;
        cout << "1.Insert addtional cash \t 2.Exit \t : ";
        cin >> confirm_fee;
      }
      if (confirm_fee == 2) {
        return;
      }
      num_of_1000 += 1;
    }
    // REQ4.3 현금/수표 accept 됨 -> 은행 계좌에 거래 결과 반영
    // 계좌 잔액에 총 예금액 더해주는 Bank 클래스 함수
    current_bank->setDeposit(this->current_acc, total_deposit);
    // REQ4.5 현금 -> ATM 기기의 available cash 늘어남
    num_of_1000 += deposit_1000;
    num_of_5000 += deposit_5000;
    num_of_10000 += deposit_10000;
    num_of_50000 += deposit_50000;
  }

  // check deposit 선택함
  else if (deposit_type == 2) {
    type = "Check Deposit";
    cout << "How much do you want to deposit?" << endl;
    cout << "Insert the amount of check : ";
    cin >> deposit_amount;
    if (deposit_amount == -1) {
      DisplayOfAccountATMEng();
      cout << "How much do you want to deposit?" << endl;
      cout << "Insert the amount of check : ";
```

```
    cin >> deposit_amount;
  }
  if (deposit_amount < -1) {
    cout << "Error : Amount can't be negative"
         << endl; // amount가 0보다 낮으면 안됨
    cout << endl;
    return;
  }
  cout << endl;
  int check_amount;
  for (int i = 0; i < deposit_amount; i++) {
    cout << "Insert the amount of money of check " << i << " : ";
    cin >> check_amount;
    if (check_amount == -1) {
      DisplayOfAccountATMEng();
      cout << "Insert the amount of money of check " << i << " : ";
      cin >> check_amount;
    }
    total_deposit += check_amount;
  }
  cout << endl;

  // REQ4.2 개수 제한 넘으면 display error message
  if (deposit_amount > check_limit) {
    cout << "[ERROR] Your deposit amount is over 50(check deposit limit)."
         << endl;
    return;
  }
  // REQ4.4 이체 수수료 부과될 수 있음 (non-primary bank로 deposit 할 때)
  if ((this->serial_type == 2) &&
      (current_acc->getBankName() != serial_primary_bank->getBankName())) {
    int confirm_fee = 2;
    cout << "You are charged for deposit fee of 1000won." << endl;
    cout << "1.Insert addtional cash \t 2.Exit \t : ";
    cin >> confirm_fee;
    if (confirm_fee == -1) {
      DisplayOfAccountATMEng();
      cout << "You are charged for deposit fee of 1000won." << endl;
      cout << "1.Insert addtional cash \t 2.Exit \t : ";
      cin >> confirm_fee;
    }
    if (confirm_fee == 2) {
      return;
    }
    num_of_1000 += 1;
  }
  // REQ4.3 현금/수표 accept됨 -> 은행 계좌에 거래 결과 반영
  // 계좌 잔액에 총 예금액 더해주는 Bank 클래스 함수
```

```
      current_bank->setDeposit(this->current_acc, total_deposit);

    }
    // REQ4.6 수표 -> available cash 변화 x

    // Exit 선택함
    else if (deposit_type == 3) {
      return;
    }
    // Exception handling
    else {
      cout << "[ERROR] You must choose from 1 to 3." << endl;
      return;
    }

    cout << "Success! You have deposited " << total_deposit
         << " won to your account." << endl;
    cout << endl;
    cout << "The remaining balance in your " << current_bank->getBankName()
         << " account " << current_acc->getAccountNum();
    cout << " is " << current_acc->getBalance() << " won" << endl;
    cout << endl;
    cout << "The remaining 1000 in ATM : " << num_of_1000 * 1000 << endl;
    cout << "The remaining 5000 in ATM : " << num_of_5000 * 5000 << endl;
    cout << "The remaining 10000 in ATM : " << num_of_10000 * 10000 << endl;
    cout << "The remaining 50000 in ATM : " << num_of_50000 * 50000 << endl;
    cout << endl;

    ATMHistory("Deposit", current_acc, nullptr, total_deposit, type);
  }
  if (this->language_you_want == 1) {
    // REQ4.1 현금/수표 입력받음
    cout << "어떤 형태로 예금을 하십니까?";
    cout << "1.화폐 \t 2.수표 \t 3. 취소" << endl;
    cin >> deposit_type;
    if (deposit_type == -1) {
      DisplayOfAccountATMKo();
      cout << "어떤 형태로 예금을 하십니까?";
      cout << "1.화폐 \t 2.수표 \t 3. 취소" << endl;
      cin >> deposit_type;
    }
    cout << endl;

    // cash deposit 선택함
    if (deposit_type == 1) {
      type = "Cash Deposit";
      cout << "예금하고자 하는 양을 각각 입력해주세요" << endl;
      cout << "천원 지폐 개수 : ";
```

```
    cin >> deposit_1000;
    if (deposit_1000 == -1) {
      DisplayOfAccountATMKo();
      cout << "예금하고자 하는 양을 각각 입력해주세요" << endl;
      cout << "천원 지폐 개수 : ";
      cin >> deposit_1000;
    }
    cout << endl;
    if (deposit_1000 < -1) { // 0 이하의 정수 받았을 때
      cout << "[오류] 수량은 0보다 작을 수 없습니다." << endl;
      return;
    }
    cout << "오천원 지폐 개수 : ";
    cin >> deposit_5000;
    if (deposit_5000 == -1) {
      DisplayOfAccountATMKo();
      cout << "예금하고자 하는 양을 각각 입력해주세요" << endl;
      cout << "오천원 지폐 개수 : ";
      cin >> deposit_5000;
    }
    cout << endl;
    if (deposit_5000 < -1) { // 0 이하의 정수 받았을 때
      cout << "[오류] 수량은 0보다 작을 수 없습니다." << endl;
      return;
    }
    cout << "만원 지폐 개수 : ";
    cin >> deposit_10000;
    if (deposit_10000 == -1) {
      DisplayOfAccountATMKo();
      cout << "예금하고자 하는 양을 각각 입력해주세요" << endl;
      cout << "만원 지폐 개수 : ";
      cin >> deposit_10000;
    }
    cout << endl;
    if (deposit_10000 < -1) { // 0 이하의 정수 받았을 때
      cout << "[오류] 수량은 0보다 작을 수 없습니다." << endl;
      cout << endl;
      return;
    }
    cout << "오만원 지폐 개수 : ";
    cin >> deposit_50000;
    if (deposit_50000 == -1) {
      DisplayOfAccountATMKo();
      cout << "예금하고자 하는 양을 각각 입력해주세요" << endl;
      cout << "오만원 지폐 개수 : ";
      cin >> deposit_50000;
    }
    cout << endl;
```

```
    if (deposit_50000 < -1) { // 0 이하의 정수 받았을 때
      cout << "[오류] 수량은 0 보다 작을 수 없습니다." << endl;
      cout << endl;
      return;
    }

    deposit_amount =
        deposit_1000 + deposit_5000 + deposit_10000 + deposit_50000;
    total_deposit = deposit_1000 * 1000 + deposit_5000 * 5000 +
                    deposit_10000 * 10000 + deposit_50000 * 50000;
    // Exceptional Handling : 이체량이 0 원이면 Error Message
    if (deposit_amount == 0) {
      cout << "[오류] 예금을 위해선 최소 천원을 입력해야 합니다." << endl;
      cout << endl;
      return;
    }
    // REQ4.2 개수 제한 넘으면 display error message
    if (deposit_amount > cash_limit) {
      cout << "[오류] 예금 가능한 최대 지폐 장 수는 총 30 장입니다." << endl;
      return;
    }

    // REQ4.4 이체 수수료 부과될 수 있음 ( non - primary bank deposit 일 때)
    if ((this->serial_type == 2) &&
        (current_acc->getBankName() != serial_primary_bank->getBankName())) {
      int confirm_fee = 2;
      cout << "예금 수수료로 1000 원이 청구됩니다." << endl;
      cout << "1.추가 현금 넣기 \t 2.취소 \t : ";
      cin >> confirm_fee;
      if (confirm_fee == -1) {
        DisplayOfAccountATMKo();
        cout << "예금 수수료로 1000 원이 청구됩니다." << endl;
        cout << "1.추가 현금 넣기 \t 2.취소 \t : ";
        cin >> confirm_fee;
      }
      if (confirm_fee == 2) {
        return;
      }
      num_of_1000 += 1;
    }
    // REQ4.3 현금/수표 accept 됨 -> 은행 계좌에 거래 결과 반영
    // 계좌 잔액에 총 예금액 더해주는 Bank 클래스 함수
    current_bank->setDeposit(this->current_acc, total_deposit);
    // REQ4.5 현금 -> ATM 기기의 available cash 늘어남
    num_of_1000 += deposit_1000;
    num_of_5000 += deposit_5000;
    num_of_10000 += deposit_10000;
    num_of_50000 += deposit_50000;
```

```
  }

  // check deposit 선택함
  else if (deposit_type == 2) {
    type = "Check Deposit";
    cout << "예금하고자 수표의 장 수를 입력해주세요." << endl;
    cin >> deposit_amount;
    if (deposit_amount == -1) {
      DisplayOfAccountATMKo();
      cout << "예금하고자 수표의 장 수를 입력해주세요." << endl;
      cin >> deposit_amount;
    }
    if (deposit_amount < -1) {
      cout << "[오류] 수량은 0 보다 작을 수 없습니다."
           << endl; // amount 가 0 보다 낮으면 안됨
      cout << endl;
      return;
    }
    cout << endl;
    int check_amount;
    for (int i = 0; i < deposit_amount; i++) {
      cout << (i + 1) << "번째 수표의 금액을 입력해주세요. : ";
      cin >> check_amount;
      if (check_amount == -1) {
        DisplayOfAccountATMKo();
        cout << (i + 1) << "번째 수표의 금액을 입력해주세요. : ";
        cin >> check_amount;
      }
      total_deposit += check_amount;
    }
    cout << endl;

    // REQ4.2 개수 제한 넘으면 display error message
    if (deposit_amount > check_limit) {
      cout << "[오류] 예금 가능한 최대 수표 장 수는 50 장입니다." << endl;
      return;
    }
    // REQ4.4 이체 수수료 부과될 수 있음 (non-primary bank 로 deposit 할 때)
    if ((this->serial_type == 2) &&
        (current_acc->getBankName() != serial_primary_bank->getBankName())) {
      int confirm_fee = 2;
      cout << "예금 수수료로 1000 원이 청구됩니다." << endl;
      cout << "1.추가 현금 넣기 \t 2.취소 \t : ";
      cin >> confirm_fee;
      if (confirm_fee == -1) {
        DisplayOfAccountATMKo();
        cout << "예금 수수료로 1000 원이 청구됩니다." << endl;
        cout << "1.추가 현금 넣기 \t 2.취소 \t : ";
```

```
          cin >> confirm_fee;
        }
        if (confirm_fee == 2) {
          return;
        }
        num_of_1000 += 1;
      }
      // REQ4.3 현금/수표 accept 됨 -> 은행 계좌에 거래 결과 반영
      // 계좌 잔액에 총 예금액 더해주는 Bank 클래스 함수
      current_bank->setDeposit(this->current_acc, total_deposit);

    }
    // REQ4.6 수표 -> available cash 변화 x

    // Exit 선택함
    else if (deposit_type == 3) {
      return;
    }
    // Exception handling
    else {
      cout << "[오류] 예금 종류 1~3 중에 선택해야 합니다.." << endl;
      return;
    }

    cout << total_deposit << "원이 계좌로 입금되었습니다." << endl;
    cout << endl;
    cout << current_bank->getBankName() << " 은행의 "
         << current_acc->getAccountNum() << "계좌에 남아있는 총 금액은 ";
    cout << current_acc->getBalance() << "원입니다." << endl;
    cout << endl;
    cout << "ATM 에 존재하는 천원권의 총 금액 : " << num_of_1000 * 1000 << endl;
    cout << "ATM 에 존재하는 오천원의 총 금액 : " << num_of_5000 * 5000 << endl;
    cout << "ATM 에 존재하는 만원권의 총 금액 : " << num_of_10000 * 10000
         << endl;
    cout << "ATM 에 존재하는 오만원의 총 금액 : " << num_of_50000 * 50000
         << endl;
    cout << endl;

    ATMHistory("Deposit", current_acc, nullptr, total_deposit, type);
  }
}

// 5. Withdrawal
void ATM::withdrawal() {
  // check 는 withdrawal 불가능
  // int withdrawal_amount; // deposit 장수
  // int total_withdrawal;  // deposit 총액
  int cash_limit = 30;
```

```
int denominationOfBills; // 지폐 종류
int wrong_input = 0;     // 틀린 입력일 때 종료를 위한 변수
int sumOfWithdrawal = 0;
int withdrawal_1000 = 0;
int withdrawal_5000 = 0;
int withdrawal_10000 = 0;
int withdrawal_50000 = 0;
int fund;
if (this->language_you_want == 2) {
  // 각 session 당 withdrawal 횟수는 3 회까지
  // !!!!!!!!!!!!!!!!!!
  if (withdrawal_count < 3) {
    // REQ5.1 An ATM shall ask a user to enter the amount of fund to withdra
    cout << "Insert the amount of cash you want to withdrawal." << endl;
    cout << "What denomination of bills would you like to withdraw? (1,000 "
            "won, 5,000 won, 10,000 won, 50,000 won)"
         << endl;
    cout << "please input 1000, 5000, 10000, 50000...: ";
    cin >> denominationOfBills;
    if (denominationOfBills == -1) {
      DisplayOfAccountATMEng();
      cout << "Insert the amount of cash you want to withdrawal." << endl;
      cout << "What denomination of bills would you like to withdraw? (1,000 "
              "won, 5,000 won, 10,000 won, 50,000 won)"
           << endl;
      cout << "please input 1000, 5000, 10000, 50000...: ";
      cin >> denominationOfBills;
    }
    cout << endl;

    if (denominationOfBills == 1000) {
      cout << "How many bills of would you like to withdraw?" << endl;
      cout << "please input 1000, 2000, 3000...: ";
      cin >> sumOfWithdrawal;
      if (sumOfWithdrawal == -1) {
        DisplayOfAccountATMEng();
        cout << "How many bills of would you like to withdraw?" << endl;
        cout << "please input 1000, 2000, 3000...: ";
        cin >> sumOfWithdrawal;
      }
      cout << endl;
      if (sumOfWithdrawal % 1000 == 0) {
        withdrawal_1000 = sumOfWithdrawal / 1000;
      } else {
        cout << "[Error] Please input the correct amount of bills that you "
                "want to withdrawal."
             << endl;
        wrong_input = 1;
```

```
    }
  } else if (denominationOfBills == 5000) {
    cout << "How many bills of would you like to withdraw?" << endl;
    cout << "please input 5000, 10000, 15000...: ";
    cin >> sumOfWithdrawal;
    if (sumOfWithdrawal == -1) {
      DisplayOfAccountATMEng();
      cout << "How many bills of would you like to withdraw?" << endl;
      cout << "please input 5000, 10000, 15000...: ";
      cin >> sumOfWithdrawal;
    }
    cout << endl;
    if (sumOfWithdrawal % 5000 == 0) {
      withdrawal_5000 = sumOfWithdrawal / 5000;
    } else {
      cout << "[Error] Please input the correct amount of bills that you "
              "want to withdrawal."
           << endl;
      wrong_input = 1;
    }
  } else if (denominationOfBills == 10000) {
    cout << "How many bills of would you like to withdraw?" << endl;
    cout << "please input 10000, 20000, 30000...: ";
    cin >> sumOfWithdrawal;
    if (sumOfWithdrawal == -1) {
      DisplayOfAccountATMEng();
      cout << "How many bills of would you like to withdraw?" << endl;
      cout << "please input 10000, 20000, 30000...: ";
      cin >> sumOfWithdrawal;
    }
    cout << endl;
    if (sumOfWithdrawal % 10000 == 0) {
      withdrawal_10000 = sumOfWithdrawal / 10000;
    } else {
      cout << "[Error] Please input the correct amount of bills that you "
              "want to withdrawal."
           << endl;
      wrong_input = 1;
    }
  } else if (denominationOfBills == 50000) {
    cout << "How many bills of would you like to withdraw?" << endl;
    cout << "please input 50000, 100000, 150000...: ";
    cin >> sumOfWithdrawal;
    if (sumOfWithdrawal == -1) {
      DisplayOfAccountATMEng();
      cout << "How many bills of would you like to withdraw?" << endl;
      cout << "please input 50000, 100000, 150000...: ";
      cin >> sumOfWithdrawal;
```

```
      }
      cout << endl;
      if (sumOfWithdrawal % 50000 == 0) {
        withdrawal_50000 = sumOfWithdrawal / 50000;
      } else {
        cout << "[Error] Please input the correct amount of bills that you "
                "want to withdrawal."
             << endl;
        wrong_input = 1;
      }
    } else {
      cout << "[Error] Please input the correct denomination of bills"
           << endl;
      wrong_input = 1;
    }

    // REQ5.2 An ATM shall display an appropriate error message if there is
    // insufficient fund in the account or insufficient cash in the ATM.
    if (num_of_1000 < withdrawal_1000 || num_of_5000 < withdrawal_5000 ||
        num_of_10000 < withdrawal_10000 || num_of_50000 < withdrawal_50000) {
      cout << "[Error] There is not enough money in the ATM" << endl;
      cout << endl;
    } // ATM에 돈이 충분하지 않다면 Error 출력
    else if (current_acc->getBalance() < sumOfWithdrawal) {
      cout << "[Error] There is not enough money in your account" << endl;
      cout << endl;
    } // REQ 5.7 The maximum amount of cash withdrawal per transaction is KRW
      // 500,000.
    else if (500000 < sumOfWithdrawal) {
      cout << "[Error] Withdrawals over 500,000 won are not allowed" << endl;
      cout << endl;
    } else if (wrong_input == 1) {
      cout << endl;
    }

    //  REQ5.3 Once the withdrawal is successful, the transaction must be
    //  reflected to the bank account
    // as well(i.e., the same amount of fund must be deducted from the
    // corresponding bank account). REQ5.5 The cash withdrawal lower available
    // cash in the ATM that can be used by other users.
    else {
      // REQ5.4 Some withdrawal fee may be charged(See REQ in System Setup).
      if (withdrawal_fee() == true) { // 수수료 낼 수 있으면
        // ATM에서 돈 빼기
        num_of_1000 = num_of_1000 - withdrawal_1000;
        num_of_5000 = num_of_5000 - withdrawal_5000;
        num_of_10000 = num_of_10000 - withdrawal_10000;
        num_of_50000 = num_of_50000 - withdrawal_50000;
```

```
        // 계좌에서 돈 빼기
        current_bank->setWithdrawal(this->current_acc, sumOfWithdrawal);
        // 계좌/ATM에 남은 금액 출력
        cout << "Success!" << endl;
        cout << endl;
        cout << "The remaining balance in your "
             << current_bank->getBankName() << " account is ";
        cout << current_acc->getBalance() << " won" << endl;
        cout << endl;
        cout << "The remaining 1000 in ATM : " << num_of_1000 * 1000 << endl;
        cout << "The remaining 5000 in ATM : " << num_of_5000 * 5000 << endl;
        cout << "The remaining 10000 in ATM : " << num_of_10000 * 10000
             << endl;
        cout << "The remaining 50000 in ATM : " << num_of_50000 * 50000
             << endl;
        cout << endl;
        ATMHistory("Withdrawal", current_acc, nullptr, sumOfWithdrawal,
                   "None");
        withdrawal_count++;
      } else { // 너 수수료 낼 돈 없어,,
        cout << "[Error] There is not enough money in your account" << endl;
        cout << endl;
      }
    }
  } else {
    cout << "[Error] If you want more withdrawal, Please restart session"
         << endl;
    cout << endl;
    return;
  }
}

if (this->language_you_want == 1) {

  // 각 session 당 withdrawal 횟수는 3회까지
  if (withdrawal_count < 3) {
    // REQ5.1 An ATM shall ask a user to enter the amount of fund to withdra
    cout << "인출하고자 하는 금액을 적으시오." << endl;
    cout << "인출하고자 하는 지폐의 종류를 적으시오. (1,000 "
            "원, 5,000원, 10,000원, 50,000원)"
         << endl;
    cout << "[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: ";
    cin >> denominationOfBills;
    if (denominationOfBills == -1) {
      DisplayOfAccountATMKo();
      cout << "인출하고자 하는 금액을 적으시오." << endl;
      cout << "인출하고자 하는 지폐의 종류를 적으시오. (1,000 "
              "원, 5,000원, 10,000원, 50,000원)"
```

```
            << endl;
      cout << "[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: ";
      cin >> denominationOfBills;
    }
    cout << endl;

    if (denominationOfBills == 1000) {
      cout << "인출하고자 하는 천원권의 총 금액을 입력하시오. " << endl;
      cout << "[주의] 1000, 2000, 3000 단위로 적으시오. ";
      cin >> sumOfWithdrawal;
      if (sumOfWithdrawal == -1) {
        DisplayOfAccountATMKo();
        cout << "인출하고자 하는 천원권의 총 금액을 입력하시오. " << endl;
        cout << "[주의] 1000, 2000, 3000 단위로 적으시오. ";
        cin >> sumOfWithdrawal;
      }
      cout << endl;
      if (sumOfWithdrawal % 1000 == 0) {
        withdrawal_1000 = sumOfWithdrawal / 1000;
      } else {
        cout << "[오류] 인출하고자 하는 단위를 위 사항에 맞게 적으시오. "

             << endl;
        wrong_input = 1;
      }
    } else if (denominationOfBills == 5000) {
      cout << "인출하고자 하는 오천원권의 총 금액을 입력하시오. " << endl;
      cout << "[주의] 5000, 10000, 15000 단위로 적으시오. ";
      cin >> sumOfWithdrawal;
      if (sumOfWithdrawal == -1) {
        DisplayOfAccountATMKo();
        cout << "인출하고자 하는 오천원권의 총 금액을 입력하시오" << endl;
        cout << "[주의] 5000, 10000, 15000 단위로 적으시오. ";
        cin >> sumOfWithdrawal;
      }
      cout << endl;
      if (sumOfWithdrawal % 5000 == 0) {
        withdrawal_5000 = sumOfWithdrawal / 5000;
      } else {
        cout << "[오류] 인출하고자 하는 단위를 위 사항에 맞게 적으시오."
             << endl;
        wrong_input = 1;
      }
    } else if (denominationOfBills == 10000) {
      cout << "인출하고자 하는 만원권의 총 금액을 입력하시오. " << endl;
      cout << "[주의] 10000, 20000, 30000 단위로 적으시오. ";
      cin >> sumOfWithdrawal;
      if (sumOfWithdrawal == -1) {
```

```
        DisplayOfAccountATMKo();
        cout << "인출하고자 하는 만원권의 총 금액을 입력하시오." << endl;
        cout << "[주의] 10000, 20000, 30000 단위로 적으시오. ";
        cin >> sumOfWithdrawal;
      }
      cout << endl;
      if (sumOfWithdrawal % 10000 == 0) {
        withdrawal_10000 = sumOfWithdrawal / 10000;
      } else {
        cout << "[오류] 인출하고자 하는 단위를 위 사항에 맞게 적으시오."
             << endl;
        wrong_input = 1;
      }
    } else if (denominationOfBills == 50000) {
      cout << "인출하고자 하는 오만원권의 총 금액을 입력하시오." << endl;
      cout << "[주의] 50000, 100000, 150000 단위로 적으시오. ";
      cin >> sumOfWithdrawal;
      if (sumOfWithdrawal == -1) {
        DisplayOfAccountATMKo();
        cout << "인출하고자 하는 오만원권의 총 금액을 입력하시오." << endl;
        cout << "[주의] 50000, 100000, 150000 단위로 적으시오. ";
        cin >> sumOfWithdrawal;
      }
      cout << endl;
      if (sumOfWithdrawal % 50000 == 0) {
        withdrawal_50000 = sumOfWithdrawal / 50000;
      } else {
        cout << "[오류] 인출하고자 하는 단위를 위 사항에 맞게 적으시오."
             << endl;
        wrong_input = 1;
      }
    } else {
      cout << "[오류] 인출하고자 하는 지폐의 종류를 위 사항에 맞게 적으시오."
           << endl;
      wrong_input = 1;
    }

    // REQ5.2 An ATM shall display an appropriate error message if there is
    // insufficient fund in the account or insufficient cash in the ATM.
    if (num_of_1000 < withdrawal_1000 || num_of_5000 < withdrawal_5000 ||
        num_of_10000 < withdrawal_10000 || num_of_50000 < withdrawal_50000) {
      cout << "[오류] ATM 에 들어있는 돈이 부족합니다. " << endl;
      cout << endl;
    } // ATM 에 돈이 충분하지 않다면 Error 출력
    else if (current_acc->getBalance() < sumOfWithdrawal) {
      cout << "[오류] 계좌에 충분한 금액이 들어있지 않습니다." << endl;
      cout << endl;
    } // REQ 5.7 The maximum amount of cash withdrawal per transaction is KRW
```

```
      // 500,000.
    else if (500000 < sumOfWithdrawal) {
      cout << "[오류] 오십만원이 넘는 인출은 불가능합니다." << endl;
      cout << endl;
    } else if (wrong_input == 1) {
      cout << endl;
    }

    //  REQ5.3 Once the withdrawal is successful, the transaction must be
    //  reflected to the bank account
    // as well(i.e., the same amount of fund must be deducted from the
    // corresponding bank account). REQ5.5 The cash withdrawal lower available
    // cash in the ATM that can be used by other users.
    else {
      // REQ5.4 Some withdrawal fee may be charged(See REQ in System Setup).
      if (withdrawal_fee() == true) { // 수수료 낼 수 있으면
        // ATM 에서 돈 빼기
        num_of_1000 = num_of_1000 - withdrawal_1000;
        num_of_5000 = num_of_5000 - withdrawal_5000;
        num_of_10000 = num_of_10000 - withdrawal_10000;
        num_of_50000 = num_of_50000 - withdrawal_50000;
        // 계좌에서 돈 빼기
        current_bank->setWithdrawal(this->current_acc, sumOfWithdrawal);
        // 계좌/ATM 에 남은 금액 출력
        cout << "성공하였습니다." << endl;
        cout << endl;
        cout << current_bank->getBankName() << "의 계좌는 ";
        cout << current_acc->getBalance() << "원이 되었습니다." << endl;
        cout << endl;
        cout << "ATM 에 존재하는 천원권의 총 금액 : " << num_of_1000 * 1000
             << endl;
        cout << "ATM 에 존재하는 오천원의 총 금액 : " << num_of_5000 * 5000
             << endl;
        cout << "ATM 에 존재하는 만원권의 총 금액 : " << num_of_10000 * 10000
             << endl;
        cout << "ATM 에 존재하는 오만원의 총 금액 : " << num_of_50000 * 50000
             << endl;
        cout << endl;
        ATMHistory("Withdrawal", current_acc, nullptr, sumOfWithdrawal,
                   "None");
        withdrawal_count++;
      } else { // 너 수수료 낼 돈 없어,,
        cout << "[오류] 계좌에 충분한 돈이 존재하지 않습니다." << endl;
        cout << endl;
      }
    }
  } else {
    cout << "[오류] 추가적으로 인출하려면 새로운 세션을 진행하세요." << endl;
```

```
      cout << endl;
      return;
    }
  }
}
bool ATM::withdrawal_fee() {
  // Withdrawal fee for a primary bank: KRW 1,000; the fee is paid from the
  // withdrawal account.
  if (current_bank == serial_primary_bank) {
    if (current_acc->getBalance() < 1000) { // 수수료가 통장에 없는 경우
      return false;
    } else {
      current_bank->setWithdrawal(this->current_acc, 1000); // 1000 원 수수료
      if (language_you_want == 2) {
        cout << "You are charged for withdrawal fee of 1000won." << endl;
      } else if (language_you_want == 1) {
        cout << "인출 수수료로 1000 원이 청구됩니다." << endl;
      }
      return true;
    }
  } else {
    if (current_acc->getBalance() < 2000) {
      return false;
    } else {
      current_bank->setWithdrawal(this->current_acc, 2000); // 2000 원 수수료
      if (language_you_want == 2) {
        cout << "You are charged for withdrawal fee of 2000won." << endl;
      } else if (language_you_want == 1) {
        cout << "인출 수수료로 2000 원이 청구됩니다." << endl;
      }
      return true;
    }
  }
}

// 6. transfer
void ATM::transfer() {
  // destination 이 보낼 account
  // source 이 현재 account
  int transfer_type;  // cash or account 이체
  int total_transfer; // 이체 될 금액 (이건 수수료 포함 X)
  string destination_acc_num;
  int transfer_fee;
  Bank *destination_bank = nullptr;
  Account *destination_acc = nullptr;

  if (this->language_you_want == 2) {
    // REQ 6.1 cash transfer / account fund transfer 중 transfer type 입력받음
```

```
    cout << "What is your transfer type? ";
    cout << "1.cash transfer\t2.account transfer\t3. Exit : ";
    cin >> transfer_type;
    if (transfer_type == -1) {
      DisplayOfAccountATMEng();
      cout << "What is your transfer type? ";
      cout << "1.cash transfer\t2.account transfer\t3. Exit : ";
      cin >> transfer_type;
    }
    cout << endl;

    if(transfer_type == 3){
      cout << endl;
      return;
    }

    // REQ 6.2 이체해주려는 destination account number 입력받음
    cout << "What is destination account number? : ";
    cin >> destination_acc_num;
    if (destination_acc_num == "-1") {
      DisplayOfAccountATMEng();
      cout << "What is destination account number? : ";
      cin >> destination_acc_num;
    }
    cout << endl;

    for (int i = 0; i < Bank_list.size(); i++) { // bank_list 가 10 개라서 이케함,
len 따로 구해도 됨
      destination_acc = Bank_list[i]->findAccount(destination_acc_num);
      if (destination_acc != nullptr) { // 찾으면 break
        destination_bank = Bank_list[i];
        break;
      }
    }
    while(destination_acc == nullptr){
    // destination account 의 account 포인터와 bank 포인터 찾음
      for (int i = 0; i < Bank_list.size(); i++) { // bank_list 가 10 개라서 이케함,
len 따로 구해도 됨
        destination_acc = Bank_list[i]->findAccount(destination_acc_num);
        if (destination_acc != nullptr) { // 찾으면 break
          destination_bank = Bank_list[i];
          break;
        }
      }
      cout << "Account with number " << destination_acc_num << " does not exist."
<< endl;
      cout << endl;
      cout << "What is destination account number? : ";
```

```
      cin >> destination_acc_num;
    }

    // cash transfer 일 때
    if (transfer_type == 1) {
      type = "Cash Transfer";
      int transfer_1000;
      int transfer_5000;
      int transfer_10000;
      int transfer_50000;
      string confirmed_fee;
      int transfer_amount;

      cout << "How much do you want to tranfer?";
      cout << "( Transfer fee of 5,000 won will be excluded from the inserted
cash. ) " << endl;
      cout << "Insert the amount of 1000 won cash : ";
      cin >> transfer_1000;
      if (transfer_1000 == -1) {
        DisplayOfAccountATMEng();
        cout << "How much do you want to tranfer? (" << endl;
        cout << "Insert the amount of 1000 won cash : ";
        cin >> transfer_1000;
      }
      cout << endl;
      cout << "Insert the amount of 5000 won cash : ";
      cin >> transfer_5000;
      if (transfer_5000 == -1) {
        DisplayOfAccountATMEng();
        cout << "Insert the amount of 5000 won cash : ";
        cin >> transfer_5000;
      }
      cout << endl;
      cout << "Insert the amount of 10000 won cash : ";
      cin >> transfer_10000;
      if (transfer_10000 == -1) {
        DisplayOfAccountATMEng();
        cout << "Insert the amount of 10000 won cash : ";
        cin >> transfer_10000;
      }
      cout << endl;
      cout << "Insert the amount of 50000 won cash : ";
      cin >> transfer_50000;
      if (transfer_50000 == -1) {
        DisplayOfAccountATMEng();
        cout << "Insert the amount of 50000 won cash : ";
        cin >> transfer_50000;
      }
```

```
      cout << endl;

      total_transfer = transfer_1000 * 1000 + transfer_5000 * 5000 +
transfer_10000 * 10000 + transfer_50000 * 50000;
      total_transfer -= 5000;
      cout << "Your total transfer fund is " << total_transfer <<" won. (If
confirmed, please press Y) : ";
      cin >> confirmed_fee;
      if (confirmed_fee == "-1") {
        DisplayOfAccountATMEng();
        cout << "Your total transfer fund is " << total_transfer <<" won. (If
confirmed, please press Y) : ";
        cin >> confirmed_fee;
      }
      if (confirmed_fee != "Y"){
        return;
      }
      cout << endl;
      // REQ 6.6. if cash transfer → available cash 늘어남
      // ATM에 돈 추가
      num_of_1000 += transfer_1000;
      num_of_5000 += transfer_5000;
      num_of_10000 += transfer_10000;
      num_of_50000 += transfer_50000;

      // 계좌로 돈 이체
      destination_bank->setTransfer(destination_acc, total_transfer);
    }
    // account transfer일 때
    else if (transfer_type == 2) {
      type = "Account Transfer";
      int check_source;
      cout << "Is your source account number " << current_acc->getAccountNum()
           << "?" << endl;
      cout << "1.Yes\t2.No : ";
      cin >> check_source;
      if (check_source == -1) {
        DisplayOfAccountATMEng();
        cout << "Is your source account number " << current_acc->getAccountNum()
             << "?" << endl;
        cout << "1.Yes\t2.No : ";
        cin >> check_source;
      }
      cout << endl;
      if (check_source == 2) {
        cout << "[ERROR] Please restart" << endl;
        return;
      }
```

```
    cout << "How much do you want to tranfer?" << endl;
    cin >> total_transfer;
    if (total_transfer == -1) {
      DisplayOfAccountATMEng();
      cout << "How much do you want to tranfer?" << endl;
      cin >> total_transfer;
    }
    cout << endl;

    // REQ 6.5. 이체 수수료 부과될 수 있음 (system setup 확인)
    bool destination_primary = (destination_bank->getBankName() ==
                                serial_primary_bank->getBankName())
                                   ? true
                                   : false;
    bool source_primary =
        (current_bank->getBankName() == serial_primary_bank->getBankName())
            ? true
            : false;
    // both primary banks
    if (destination_primary && source_primary) {
      transfer_fee = 2000;
    }
    // primary and non-primary banks
    else if (destination_primary || source_primary) {
      transfer_fee = 3000;
    }
    // both non-primary banks
    else {
      transfer_fee = 4000;
    }
    cout << "The transfer fee of " << transfer_fee
         << " won will be excluded from your account." << endl;

    //(REQ 6.7.) 이체 끝나면 잔액 변화 (있다면 source account,) destination
    // account 에 반영됨
    // -> the same amount of fund must be deducted from the source bank
    // account,and then added to the destination bank account
    destination_bank->setTransfer(destination_acc, total_transfer);
    current_bank->setWithdrawal(current_acc, total_transfer + transfer_fee);
  }

  // Exit 선택함
  else if (transfer_type == 3) {
    return;
  }
  // Exception handling
  else {
    cout << "[ERROR] You must choose from 1 to 3." << endl;
```

```
    transfer();
    return;
  }

  cout << "Success! You have transferred " << total_transfer
       << " won to account " << destination_acc_num << endl;
  if (transfer_type ==
      2) { // account transfer 일 경우에 transfer 마치고 다음 것 출력
    cout << "with account transfer." << endl;
    cout << "The remaining balance of the source account "
         << current_bank->getBankName() << " account ";
    cout << current_acc->getAccountNum() << " is "
         << current_acc->getBalance() << " won" << endl;
    cout << "The remaining balance of the destinaton account "
         << destination_bank->getBankName() << " account ";
    cout << destination_acc->getAccountNum() << " is "
         << destination_acc->getBalance() << " won" << endl;
    ATMHistory("Transfer", current_acc, destination_acc, total_transfer,
               type);
  } else if (transfer_type ==
             1) { // cash transfer 일 경우에 transfer 마치고 다음 것 출력
    cout << "with cash transfer." << endl;
    cout << "The remaining 1000 in ATM : " << num_of_1000 * 1000 << endl;
    cout << "The remaining 5000 in ATM : " << num_of_5000 * 5000 << endl;
    cout << "The remaining 10000 in ATM : " << num_of_10000 * 10000 << endl;
    cout << "The remaining 50000 in ATM : " << num_of_50000 * 50000 << endl;
    cout << destination_acc->getAccountNum() << " is "
         << destination_acc->getBalance() << " won" << endl;
    cout << "The remaining balance in your "
         << destination_bank->getBankName() << " account ";
    cout << destination_acc->getAccountNum() << " is "
         << destination_acc->getBalance() << " won" << endl;
    ATMHistory("Transfer", nullptr, destination_acc, total_transfer, type);
  }
  cout << "Transfer finished." << endl;
}

if (this->language_you_want == 1) {
  // REQ 6.1 cash transfer / account fund transfer 중 transfer type 입력받음
  cout << "이체 종류를 고르시오. ";
  cout << "1.현금 이체\t2.계좌 이체\t3. 취소 ";
  cin >> transfer_type;
  if (transfer_type == -1) {
    DisplayOfAccountATMKo();
    cout << "1.현금 이체\t2.계좌 이체\t3. 취소 ";
    cin >> transfer_type;
  }
  cout << endl;
```

```
    if (transfer_type == 3){
      return;
    }

    // REQ 6.2 이체해주려는 destination account number 입력받음
    cout << "돈을 보내려는 계좌를 고르시오. ";
    cin >> destination_acc_num;
    if (destination_acc_num == "-1") {
      DisplayOfAccountATMKo();
      cout << "돈을 보내려는 계좌를 고르시오. ";
      cin >> destination_acc_num;
    }
    cout << endl;
    for (int i = 0; i < Bank_list.size(); i++) { // bank_list 가 10 개라서 이케함,
len 따로 구해도 됨
      destination_acc = Bank_list[i]->findAccount(destination_acc_num);
      if (destination_acc != nullptr) { // 찾으면 break
        destination_bank = Bank_list[i];
        break;
      }
    }
    while(destination_acc == nullptr){
    // destination account 의 account 포인터와 bank 포인터 찾음
      for (int i = 0; i < Bank_list.size(); i++) { // bank_list 가 10 개라서 이케함,
len 따로 구해도 됨
        destination_acc = Bank_list[i]->findAccount(destination_acc_num);
        if (destination_acc != nullptr) { // 찾으면 break
          destination_bank = Bank_list[i];
          break;
        }
      }
      cout << "번호 " << destination_acc_num << "를 갖는 계좌가 존재하지 않습니다."
<< endl;
      cout << endl;
      cout << "돈을 보내려는 계좌를 다시 고르시오.";
      cin >> destination_acc_num;
    }


    for (int i = 0; i < Bank_list.size();
         i++) { // bank_list 가 10 개라서 이케함, len 따로 구해도 됨
      destination_acc = Bank_list[i]->findAccount(destination_acc_num);
      if (destination_acc != nullptr) { // 찾으면 break
        destination_bank = Bank_list[i];
        break;
      }
    }
```

```
    // cash transfer 선택함
    // REQ 6.3. if cash transfer → 수수료 포함한 현금 입력받고 이체 금액
    // 맞는지 확인 후 수수료 제외 나머지 금액 이체 Cash transfer fee to any
    // bank type: KRW 5,000; the fee is paid by inserting additional cash.
    if (transfer_type == 1) {
      type = "Cash Transfer";
      int transfer_1000;
      int transfer_5000;
      int transfer_10000;
      int transfer_50000;
      string confirmed_fee;
      int transfer_amount;

      cout << "이체하고자 하는 금액을 적으시오." << endl;
      cout << "( 이체 수수료 5,000원이 입금된 금액으로부터 제외될 것입니다. )" <<
endl;
      cout << "1,000원 지폐 개수 : ";
      cin >> transfer_1000;
      if (transfer_1000 == -1) {
        DisplayOfAccountATMKo();
        cout << "이체하고자 하는 금액을 적으시오." << endl;
        cout << "( 이체 수수료 5,000원이 입금된 금액으로부터 제외될 것입니다. )" <<
endl;
        cout << "1,000원 지폐 개수 : ";
        cin >> transfer_1000;
      }
      cout << endl;
      cout << "5,000원 지폐 개수 : ";
      cin >> transfer_5000;
      if (transfer_5000 == -1) {
        DisplayOfAccountATMKo();
        cout << "5,000원 지폐 개수 : ";
        cin >> transfer_5000;
      }
      cout << endl;
      cout << "10,000원 지폐 개수 : ";
      cin >> transfer_10000;
      if (transfer_10000 == -1) {
        DisplayOfAccountATMKo();
        cout << "10,000원 지폐 개수 : ";
        cin >> transfer_10000;
      }
      cout << endl;
      cout << "50,000원 지폐 개수 : ";
      cin >> transfer_50000;
      if (transfer_50000 == -1) {
        DisplayOfAccountATMKo();
        cout << "50,000원 지폐 개수 : ";
```

```
        cin >> transfer_50000;
      }
      cout << endl;


      total_transfer = transfer_1000 * 1000 + transfer_5000 * 5000 +
transfer_10000 * 10000 + transfer_50000 * 50000;
      total_transfer -= 5000;
      cout << "총 이체 금액은 " << total_transfer <<" 원 입니다. (동의할 경우, Y를
입력해주세요.) : ";
      cin >> confirmed_fee;
      if (confirmed_fee == "-1") {
        DisplayOfAccountATMEng();
        cout << "Your total transfer fund is " << total_transfer <<" won. (If
confirmed, please press Y) : ";
        cin >> confirmed_fee;
      }
      if (confirmed_fee != "Y"){
        return;
      }
      cout << endl;
      // REQ 6.6. if cash transfer → available cash 늘어남
      // ATM에 돈 추가
      num_of_1000 += transfer_1000;
      num_of_5000 += transfer_5000;
      num_of_10000 += transfer_10000;
      num_of_50000 += transfer_50000;

      // 계좌로 돈 이체
      destination_bank->setTransfer(destination_acc, total_transfer);
    } else if (transfer_type == 2) { // account transfer
      /*    source account number 새로 입력받는가 or 현재 카드 사용할것인지
      물어보고 넘어가는가????????? double source_acc_num; Account* source_acc;
      Bank* source_bank;
      */
      type = "Account Transfer";
      int check_source;
      cout << "돈을 빼려는 계좌가 " << current_acc->getAccountNum()
           << "가 맞습니까?" << endl;
      cout << "숫자를 입력해주세요 -> 1.네\t2.아니요 : ";
      cin >> check_source;
      if (check_source == -1) {
        DisplayOfAccountATMKo();
        cout << "돈을 빼려는 계좌가 " << current_acc->getAccountNum()
             << "가 맞습니까?" << endl;
        cout << "숫자를 입력해주세요 -> 1.네\t2.아니요 : ";
        cin >> check_source;
      }
```

```
    cout << endl;
    if (check_source == 2) {
      cout << "오류 : 다시 시작하세요 " << endl;
      return;
    }
    cout << "이체를 위한 금액을 적어주세요." << endl;
    cin >> total_transfer;
    if (total_transfer == -1) {
      DisplayOfAccountATMKo();
      cout << "이체를 위한 금액을 적어주세요." << endl;
      cin >> total_transfer;
    }
    cout << endl;

    // REQ 6.5. 이체 수수료 부과될 수 있음 (system setup 확인)
    bool destination_primary = (destination_bank->getBankName() ==
                                serial_primary_bank->getBankName())
                                   ? true
                                   : false;
    bool source_primary =
        (current_bank->getBankName() == serial_primary_bank->getBankName())
            ? true
            : false;
    if (destination_primary && source_primary) { // both primary banks
      transfer_fee = 2000;
    } else if (destination_primary ||
               source_primary) { // primary and non-primary banks
      transfer_fee = 3000;
    } else { // both non-primary banks
      transfer_fee = 4000;
    }
    cout << "수수료 " << transfer_fee << " 원이 해당 계좌에서 인출됩니다."
         << endl;

    //(REQ 6.7.) 이체 끝나면 잔액 변화 (있다면 source account,) destination
    // account 에 반영됨
    // -> the same amount of fund must be deducted from the source bank
    // account,and then added to the destination bank account
    destination_bank->setTransfer(destination_acc, total_transfer);
    current_bank->setWithdrawal(current_acc, total_transfer + transfer_fee);
  }

  // Exit 선택함
  else if (transfer_type == 3) {
    return;
  }
  // Exception handling
  else {
```

```
      cout << "[오류] 1~3 중에서 선택해야합니다." << endl;
      transfer();
      return;
    }

    cout << "성공적으로 " << total_transfer
         << "원을 다음 도착 계좌로 이체하였습니다 -> " << destination_acc_num;
    cout << endl;
    if (transfer_type ==
        2) { // account transfer 일 경우에 transfer 마치고 다음 것 출력
      cout << "계좌 이체 결과로," << endl;
      cout << "출발 계좌에 존재하는 금액은 " << current_bank->getBankName()
           << " 계좌, ";
      cout << current_acc->getAccountNum() << " 에서 "
           << current_acc->getBalance() << "원입니다." << endl;
      cout << "도착 계좌에 존재하는 금액은 " << destination_bank->getBankName()
           << " 계좌, ";
      cout << destination_acc->getAccountNum() << " 에서 "
           << destination_acc->getBalance() << "원입니다." << endl;
      ATMHistory("Transfer", current_acc, destination_acc, total_transfer,
                 type);
    } else if (transfer_type ==
               1) { // cash transfer 일 경우에 transfer 마치고 다음 것 출력
      cout << "현금 이체 결과로," << endl;
      cout << "ATM 에 존재하는 천원권의 총 금액 : " << num_of_1000 * 1000
           << endl;
      cout << "ATM 에 존재하는 오천원의 총 금액 : " << num_of_5000 * 5000
           << endl;
      cout << "ATM 에 존재하는 만원권의 총 금액 : " << num_of_10000 * 10000
           << endl;
      cout << "ATM 에 존재하는 오만원의 총 금액 : " << num_of_50000 * 50000
           << endl;
      cout << destination_acc->getAccountNum() << " 계좌 잔액은  "
           << destination_acc->getBalance() << " 원 입니다." << endl;
      ATMHistory("Transfer", nullptr, destination_acc, total_transfer, type);
    }
    cout << "이체 성공." << endl;
  }
}

// 7. Display of Transaction History (Admin Menu)
/*
o An administrator of an ATM may request an ATM to display the whole
transaction histories of all users from the beginning of the system start. o
(REQ7.1) When a session is started by an admin by inserting an admin card
(See REQ in System Setup), an ATM displays a menu of “Transaction History”
only. o (REQ7.2) When the “Transaction History” menu is selected, an ATM
display the information of all transactions from all users from the
```

```
beginning of the system start.
- Transaction ID, Card Number, Transaction Types, Amount, other
transaction-specific information
- Each transaction may have different types of information, so they need to
be appropriately displayed (e.g., a deposit transaction does not have the
source account information in a transfer transaction). o (REQ7.3) The
“Transaction History” information shall be outputted to the external file
(e.g., txt
*/
void ATM::PrintHistory(int session) {
  if (this->language_you_want == 2) {
    string printhistory;
    for (int i = session; i < trans; i++) {
      if (SessionSummary[i][1] == "Deposit" and
          SessionSummary[i][8] == "Cash Deposit") {
        printhistory = "Transaction ID: " + SessionSummary[i][0] +
                       "| Cash Deposit | Bank: " + SessionSummary[i][2] +
                       "| Account Number: " + SessionSummary[i][3] +
                       " / Deposit Amount: " + SessionSummary[i][6];

      }

      else if (SessionSummary[i][1] == "Deposit" and
               SessionSummary[i][8] == "Check Deposit") {
        printhistory = "Transaction ID: " + SessionSummary[i][0] +
                       "| Check Deposit | Bank: " + SessionSummary[i][2] +
                       "| Account Number: " + SessionSummary[i][3] +
                       " / Deposit Amount: " + SessionSummary[i][6];
      } else if (SessionSummary[i][1] == "Withdrawal") {

        printhistory = "Transaction ID: " + SessionSummary[i][0] +
                       "| Withdrawal | Bank: " + SessionSummary[i][2] +
                       "| Account Number: " + SessionSummary[i][3] +
                       " / Withdrawal Amount: " + SessionSummary[i][6];
      }

      else if (SessionSummary[i][1] == "Transfer" and
               SessionSummary[i][8] == "Cash Transfer") {
        printhistory = "Transaction ID: " + SessionSummary[i][0] +
                       "| Cash Transfer | Bank: " + SessionSummary[i][2] +
                       " -> " + SessionSummary[i][4] +
                       "| Account Number: " + SessionSummary[i][3] +
                       "| Destination Account Number: " + SessionSummary[i][5] +
                       " / Transfer Amount: " + SessionSummary[i][6];
      } else if (SessionSummary[i][1] == "Transfer" and
                 SessionSummary[i][8] == "Account Transfer") {
        printhistory = "Transaction ID: " + SessionSummary[i][0] +
                       "| Account Transfer | Bank: " + SessionSummary[i][2] +
```

```
                    " -> " + SessionSummary[i][4] +
                    "| Account Number: " + SessionSummary[i][3] +
                    "| Destination Account Number: " + SessionSummary[i][5] +
                    " / Transfer Amount: " + SessionSummary[i][6];
    }
    cout << printhistory << endl;
  }
}
if (this->language_you_want == 1) {
  string printhistory;
  for (int i = session; i < trans; i++) {
    if (SessionSummary[i][1] == "Deposit" and
        SessionSummary[i][8] == "Cash Deposit") {
      printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                    "| 현금 예금 | 은행: " + SessionSummary[i][2] +
                    "| 계좌번호 : " + SessionSummary[i][3] +
                    " / 예금 금액: " + SessionSummary[i][6];
    }
    if (SessionSummary[i][1] == "Deposit" and
        SessionSummary[i][8] == "Check Deposit") {
      printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                    "| 수표 예금 | 은행: " + SessionSummary[i][2] +
                    "| 계좌번호 : " + SessionSummary[i][3] +
                    " / 예금 금액: " + SessionSummary[i][6];
    } else if (SessionSummary[i][1] == "Withdrawal") {

      printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                    "| 인출 | 은행: " + SessionSummary[i][2] +
                    "| 계좌번호 : " + SessionSummary[i][3] +
                    " / 인출 금액: " + SessionSummary[i][6];
    }

    else if (SessionSummary[i][1] == "Transfer" and
             SessionSummary[i][8] == "Cash Transfer") {
      printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                    "| 현금 이체 | 은행: " + SessionSummary[i][2] + " -> " +
                    SessionSummary[i][4] +
                    "| 계좌번호 : " + SessionSummary[i][3] +
                    "| 도착 계좌번호 : " + SessionSummary[i][5] +
                    " / 이체 금액: " + SessionSummary[i][6];
    } else if (SessionSummary[i][1] == "Transfer" and
               SessionSummary[i][8] == "Check Transfer") {
      printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                    "| 계좌 이체 | 은행: " + SessionSummary[i][2] + " -> " +
                    SessionSummary[i][4] +
                    "| 계좌번호 : " + SessionSummary[i][3] +
                    "| 도착 계좌번호 : " + SessionSummary[i][5] +
                    " / 이체 금액: " + SessionSummary[i][6];
```

```
      }
      cout << printhistory << endl;
    }
  }
}

void ATM::admin() {
  if (this->language_you_want == 2) {
    int select;
    string printhistory;
    cout << "Please select your transaction" << endl;
    cout << "1. Transaction History: " << endl;
    cin >> select;
    if (select == 1) {
      ofstream letsgo;
      letsgo.open("ATM_Transaction.txt", ios::trunc);
      for (int i = 0; i < trans; i++) {
        if (SessionSummary[i][1] == "Deposit" and
            SessionSummary[i][8] == "Cash Deposit") {
          printhistory = "Transaction ID: " + SessionSummary[i][0] +
                         "| Cash Deposit | Bank: " + SessionSummary[i][2] +
                         "| Account Number: " + SessionSummary[i][3] +
                         " / Deposit Amount: " + SessionSummary[i][6];

        }

        else if (SessionSummary[i][1] == "Deposit" and
                 SessionSummary[i][8] == "Check Deposit") {
          printhistory = "Transaction ID: " + SessionSummary[i][0] +
                         "| Check Deposit | Bank: " + SessionSummary[i][2] +
                         "| Account Number: " + SessionSummary[i][3] +
                         " / Deposit Amount: " + SessionSummary[i][6];
        } else if (SessionSummary[i][1] == "Withdrawal") {

          printhistory = "Transaction ID: " + SessionSummary[i][0] +
                         "| Withdrawal | Bank: " + SessionSummary[i][2] +
                         "| Account Number: " + SessionSummary[i][3] +
                         " / Withdrawal Amount: " + SessionSummary[i][6];
        }

        else if (SessionSummary[i][1] == "Transfer" and
                 SessionSummary[i][8] == "Cash Transfer") {
          printhistory =
              "Transaction ID: " + SessionSummary[i][0] +
              "| Cash Transfer | Bank: " + SessionSummary[i][2] + " -> " +
              SessionSummary[i][4] +
              "| Account Number: " + SessionSummary[i][3] +
              "| Destination Account Number: " + SessionSummary[i][5] +
```

```
            " / Transfer Amount: " + SessionSummary[i][6];
      } else if (SessionSummary[i][1] == "Transfer" and
                 SessionSummary[i][8] == "Account Transfer") {
        printhistory =
            "Transaction ID: " + SessionSummary[i][0] +
            "| Account Transfer | Bank: " + SessionSummary[i][2] + " -> " +
            SessionSummary[i][4] +
            "| Account Number: " + SessionSummary[i][3] +
            "| Destination Account Number: " + SessionSummary[i][5] +
            " / Transfer Amount: " + SessionSummary[i][6];
      }
      cout << printhistory << endl;
      letsgo << printhistory << endl;
    }
    letsgo << endl;
    letsgo.close();
  }
  return;
}

if (this->language_you_want == 1) {
  int select;
  string printhistory;
  cout << "거래를 선택해주세요." << endl;
  cout << "1. 거래 내역: " << endl;
  cin >> select;
  if (select == 1) {
    ofstream letsgo;
    letsgo.open("ATM_Transaction.txt", ios::trunc);
    for (int i = 0; i < trans; i++) {
      if (SessionSummary[i][1] == "Deposit" and
          SessionSummary[i][8] == "Cash Deposit") {
        printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                       "| 현금 예금 | 은행: " + SessionSummary[i][2] +
                       "| 계좌번호 : " + SessionSummary[i][3] +
                       " / 예금 금액: " + SessionSummary[i][6];
      }
      if (SessionSummary[i][1] == "Deposit" and
          SessionSummary[i][8] == "Check Deposit") {
        printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                       "| 수표 예금 | 은행: " + SessionSummary[i][2] +
                       "| 계좌번호 : " + SessionSummary[i][3] +
                       " / 예금 금액: " + SessionSummary[i][6];
      } else if (SessionSummary[i][1] == "Withdrawal") {

        printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                       "| 인출 | 은행: " + SessionSummary[i][2] +
                       "| 계좌번호 : " + SessionSummary[i][3] +
```

```
                        " / 인출 금액: " + SessionSummary[i][6];
        }

        else if (SessionSummary[i][1] == "Transfer" and
                 SessionSummary[i][8] == "Cash Transfer") {
          printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                        "| 현금 이체 | 은행: " + SessionSummary[i][2] +
                        " -> " + SessionSummary[i][4] +
                        "| 계좌번호 : " + SessionSummary[i][3] +
                        "| 도착 계좌번호 : " + SessionSummary[i][5] +
                        " / 이체 금액: " + SessionSummary[i][6];
        } else if (SessionSummary[i][1] == "Transfer" and
                   SessionSummary[i][8] == "Account Transfer") {
          printhistory = "거래 고유 번호: " + SessionSummary[i][0] +
                        "| 계좌 이체 | 은행: " + SessionSummary[i][2] +
                        " -> " + SessionSummary[i][4] +
                        "| 계좌번호 : " + SessionSummary[i][3] +
                        "| 도착 계좌번호 : " + SessionSummary[i][5] +
                        " / 이체 금액: " + SessionSummary[i][6];
        }
        cout << printhistory << endl;
        letsgo << printhistory << endl;
      }

      letsgo << endl;
      letsgo.close();
    }
  }
}

void ATM::DisplayOfAccountATMEng() {
  cout << endl;
  cout << "==== all ATM, accounts info ====" << endl;
  cout << endl;
  for (int i = 0; i < Bank::ATM_list.size(); i++) {
    cout << "ATM [SN: " << Bank::ATM_list[i]->serial_num;
    int remaining_cash = Bank::ATM_list[i]->num_of_1000 * 1000 +
                         Bank::ATM_list[i]->num_of_5000 * 5000 +
                         Bank::ATM_list[i]->num_of_10000 * 10000 +
                         Bank::ATM_list[i]->num_of_50000 * 50000;
    cout << "] remaining cash: " << remaining_cash;
    cout << "(1000: " << Bank::ATM_list[i]->num_of_1000;
    cout << ", 5000: " << Bank::ATM_list[i]->num_of_5000;
    cout << ", 10000: " << Bank::ATM_list[i]->num_of_10000;
    cout << ", 50000: " << Bank::ATM_list[i]->num_of_50000;
    cout << ")";
    cout << endl;
  }
```

```
  for (int k = 0; k < Bank::Bank_list.size(); k++) {
    for (int j = 0; j < Bank_list[k]->Account_list.size(); j++) {
      cout << "Account [Bank: " << Bank::Bank_list[k]->getBankName()
           << ", No: ";
      cout << Bank::Bank_list[k]->Account_list[j]->getAccountNum()
           << ", Owner: ";
      cout << Bank::Bank_list[k]->Account_list[j]->getName();
      cout << "] balance: " << Bank::Bank_list[k]->Account_list[j]->getBalance()
           << endl;
      cout << endl;
    }
  }
  cout << "================================" << endl;
  cout << endl;
}

void ATM::DisplayOfAccountATMKo() {
  cout << endl;
  cout << "==== 모든 ATM, accounts 의 정보 ====" << endl;
  cout << endl;
  for (int i = 0; i < Bank::ATM_list.size(); i++) {
    cout << "ATM [고유 ATM 번호: " << Bank::ATM_list[i]->serial_num;
    int remaining_cash = Bank::ATM_list[i]->num_of_1000 * 1000 +
                         Bank::ATM_list[i]->num_of_5000 * 5000 +
                         Bank::ATM_list[i]->num_of_10000 * 10000 +
                         Bank::ATM_list[i]->num_of_50000 * 50000;
    cout << "] ATM 에 남아있는 잔액: " << remaining_cash;
    cout << "(1000: " << Bank::ATM_list[i]->num_of_1000;
    cout << "장 , 5000: " << Bank::ATM_list[i]->num_of_5000;
    cout << "장 , 10000: " << Bank::ATM_list[i]->num_of_10000;
    cout << "장 , 50000: " << Bank::ATM_list[i]->num_of_50000;
    cout << "장)";
    cout << endl;
  }
  for (int k = 0; k < Bank::Bank_list.size(); k++) {
    for (int j = 0; j < Bank_list[k]->Account_list.size(); j++) {
      cout << "계좌 [은행: " << Bank::Bank_list[k]->getBankName()
           << ", 계좌 번호: ";
      cout << Bank::Bank_list[k]->Account_list[j]->getAccountNum()
           << ", 계정주 : ";
      cout << Bank::Bank_list[k]->Account_list[j]->getName();
      cout << "] 계좌에 잔액: "
           << Bank::Bank_list[k]->Account_list[j]->getBalance() << endl;
      cout << endl;
    }
  }
  cout << "================================" << endl;
  cout << endl;
```

```
}
```

4. main() function and other functions

system 의 동작을 위한 *main()*과 그 외 함수들의 implementation 이다.

*Bank *findBank(string bank_name, vector<Bank *> Bank_list)*는 bank pointer 를 찾기 위한 함수이다. 해당 bank 가 존재하는지 확인할 때 사용하는 함수이다.

```
Bank *findBank(string bank_name, vector<Bank *> Bank_list) {
  Bank *bank_ptr = nullptr;
  for (int i = 0; i < Bank_list.size(); i++) {
    if (bank_name == Bank_list[i]->getBankName()) {
      bank_ptr = Bank_list[i];
      break;
    }
  }
  return bank_ptr;
}
```

*Bank *findATM(string ATM_id, vector<ATM *> ATM_list)*는 ATM pointer 를 찾기 위한 함수이다. 해당 ATM 이 존재하는지 확인할 때 사용하는 함수이다.

```
ATM *findATM(int ATM_id, vector<ATM *> ATM_list) {
  ATM *ATM_ptr = nullptr;
  for (int i = 0; i < ATM_list.size(); i++) {
    if (ATM_id == ATM_list[i]->serial_num) {
      ATM_ptr = ATM_list[i];
      break;
    }
  }
  return ATM_ptr;
}
```

*void Bank_Creation()*함수는 bank construct 를 생성하는 함수이다. *main()*에서 실행된다.

```
void Bank_Creation() {
  string bank_name;
  cout << "Insert the bank name: ";
  cin >> bank_name;
  Bank *new_bank = new Bank(bank_name);
}
```

*void Account_Creation()*함수는 account construct 를 생성하는 함수이다. *main()*에서 실행된다.

```
void Account_Creation() {
  string account_owner;
  string account_bank_name;
  Bank *account_bank_ptr;
  string account_id;
  int account_pw;
  int account_balance;
  cout << "Insert the account's bank name : ";
  cin >> account_bank_name;
  account_bank_ptr = findBank(
      account_bank_name, Bank::Bank_list); // account_bank_ptr 찾아야함 !!!!
  while (account_bank_ptr == nullptr) {
    cout << "Bank " << account_bank_name << " does not exist." << endl;
    cout << "Insert the account's bank name again : ";
    cin >> account_bank_name;
    account_bank_ptr = findBank(account_bank_name, Bank::Bank_list);
  }
  cout << "Insert the account's owner name : ";
  cin >> account_owner;
  cout << "Insert the account number : ";
  cin >> account_id;
  cout << "Insert the account password : ";
  cin >> account_pw;
  cout << "Insert the account balance : ";
  cin >> account_balance;
  Account *new_account = new Account(account_bank_name, account_owner,
                                     account_id, account_pw, account_balance);
  account_bank_ptr->setAccount(new_account);
  cout << endl;
  cout << "Bank " << account_bank_name
       << "'s account list is updated as following." << endl;
  account_bank_ptr->printAllAccounts();
}
```

*void ATM_Creation()*함수는 ATM construct 를 생성하는 함수이다. *main()*에서 실행된다.

```
void ATM_Creation() {
  string ATM_bank_name;
  Bank *ATM_bank_ptr;
  int ATM_id;
  int ATM_type;
  int ATM_lang;
  int ATM_1000;
  int ATM_5000;
  int ATM_10000;
  int ATM_50000;
```

```
  cout << "Insert the ATM's primary bank name : ";
  cin >> ATM_bank_name;
  ATM_bank_ptr =
      findBank(ATM_bank_name, Bank::Bank_list); // ATM_bank_ptr 찾아야함 !!!!
  while (ATM_bank_ptr == nullptr) {
    cout << "Bank " << ATM_bank_name << " does not exist." << endl;
    cout << "Insert the ATM's primary bank name again : ";
    cin >> ATM_bank_name;
    ATM_bank_ptr = findBank(ATM_bank_name, Bank::Bank_list);
  }
  cout << "Insert the ATM's serial number : ";
  cin >> ATM_id;
  cout << "Insert the ATM type : " << endl;
  cout << "1.Single Bank\t2.Multi-Bank" << endl;
  cin >> ATM_type;
  if(ATM_type != 1 && ATM_type != 2){
    cout << "You must enter 1 or 2." << endl;
    return;
  }
  cout << "Insert the ATM language type : " << endl;
  cout << "1.Unillingual\t2.Bilingual" << endl;
  cin >> ATM_lang;
  if(ATM_lang != 1 && ATM_lang != 2){
    cout << "You must enter 1 or 2." << endl;
    return;
  }
  cout << "Insert the number of 1000 won cash in the ATM : ";
  cin >> ATM_1000;
  cout << "Insert the number of 5000 won cash in the ATM : ";
  cin >> ATM_5000;
  cout << "Insert the number of 10000 won cash in the ATM : ";
  cin >> ATM_10000;
  cout << "Insert the number of 50000 won cash in the ATM : ";
  cin >> ATM_50000;
  ATM *new_ATM = new ATM(ATM_bank_ptr, ATM_id, ATM_type, ATM_lang, ATM_1000,
                         ATM_5000, ATM_10000, ATM_50000, Bank::Bank_list);
}
```

*main()*이다. 여기서 모든 bank, account, ATM 생성을 진행한다. 또한 ATM 사용을 진행한다.

```
int main() {
  // Bank 들 생성함
  int create_bank = 1;
  cout << "--Start making Banks--" << endl;
  while (create_bank == 1) {
    Bank_Creation();
    cout << endl;
    cout << "Do you want to create another bank?" << endl;
```

```
  cout << "1.Yes\t2.No" << endl;
  cin >> create_bank;
}
cout << "--Finish making Banks--" << endl << endl;

// Account 들 생성함
int create_account = 1;
cout << "--Start making Accounts--" << endl;
while (create_account == 1) {
  Account_Creation();
  cout << endl;
  cout << "Do you want to create another account?" << endl;
  cout << "1.Yes\t2.No" << endl;
  cin >> create_account;
}
cout << "--Finish making Accounts--" << endl << endl;

// ATM 들 생성함
int create_ATM = 1;
cout << "--Start making ATMs--" << endl;
while (create_ATM == 1) {
  ATM_Creation();
  cout << endl;
  cout << "Do you want to create another ATM?" << endl;
  cout << "1.Yes\t2.No" << endl;
  cin >> create_ATM;
}
cout << "--Finish making ATMs--" << endl << endl;

vector<Bank *> allBanks = Bank::Bank_list;
vector<ATM *> allATMs = Bank::ATM_list;

// Bank 생성 결과 확인함
cout << "num of banks: " << allBanks.size() << endl;
for (int i = 0; i < allBanks.size(); i++) {
  cout << "Bank #" << (i + 1) << " : " << allBanks[i]->getBankName() << endl;
}
cout << endl;

// ATM 생성 결과 확인함
cout << "num of ATMs: " << allATMs.size() << endl;
for (int i = 0; i < allATMs.size(); i++) {
  cout << "ATM #" << (i + 1) << " : " << allATMs[i]->serial_num << endl;
}
cout << endl;

// Use ATMs
int start_session;
```

```
  cout << "Would you like to use a ATM machine?" << endl;
  cout << "1.Yes\t2.No" << endl;
  cin >> start_session;
  while (start_session == 1) {
    int ATM_id;
    ATM *ATM_ptr;
    cout << "Insert the serial number of the ATM you want to use : ";
    cin >> ATM_id;
    ATM_ptr = findATM(ATM_id, allATMs); // ATMs 에서 A
    while (ATM_ptr == nullptr) {
      cout << "ATM with serial number " << ATM_id << " does not exist." << endl;
      cout << "Insert the serial number of the ATM again : ";
      cin >> ATM_id;
      ATM_ptr = findATM(ATM_id, allATMs);
    }
    ATM_ptr->setLanguage();
    cout << "Your session will be started." << endl;
    ATM_ptr->StartSession();
    cout << "Would you like to use a ATM machine?" << endl;
    cout << "1.Yes\t2.No" << endl;
    cin >> start_session;
  }
  return 0;
}
```

D. Instruction to run source code

Source code 실행을 위한 instruction 은 다음과 같다.

i) Set up Banks, Accounts, and ATMs (in main())

1. Bank_Creation()
    a. insert Bank_name
2. Account_Creation()
    a. insert Bank_name
    (if bank for Bank_name doesn't exist) reinsert Bank_name
    b. insert owner's name
    c. insert account number
    d. insert password
    e. insert balance
3. ATM_Creation()
    a. insert primary bank Bank_name
    (if bank for Bank_name doesn't exist) reinsert Bank_name
    b. insert ATM serial number
    c. insert ATM type (single/multi)
    d. insert ATM language type (unilingual/bilingual)
    e. insert ATM's available cash (1000won, 5000won, 10000won, 50000won)
4. use ATM

a. insert whether you want to start a ATM session
b. insert the ATM serial number
c. setLanguage()
(if ATM_language_type is bilingual) choose language (Korean/English)
d. continue to StartSession()

ii) ATM sessions (in StartSession())

1. insert card(=Account_num)
(if Account_num is Admin_id) continue to Admin()
    a. Authorization()
    (if card is invalid) authorization fail, EndSession
    (if incorrect password 3 times) authorization fail, EndSession
    b. continue to Transactions

2. Transactions
    a. Deposit()
        i. Cash deposit
            1. insert deposit amount (1000won, 5000won, 10000won, 50000won)
            (if deposit fee charged) insert an additional 1000won cash
        ii. Check deposit
            1. insert check amount
            2. insert the fund for each check
            (if deposit fee charged) insert an additional 1000won cash
    b. Withdrawal()
        i. insert withdrawal cash type (1000won / 5000won / 10000won / 50000won)
        ii. insert withdrawal amount
    c. Transfer()
        i. Cash transfer
            1. insert destination account number
            2. insert transfer amount (1000won, 5000won, 10000won, 50000won)
            3. insert transfer fee of 5000won
        ii. Account transfer
            1. insert destination account number
            2. check if the current card account is the source account
            3. insert transfer amount
            4. notify transfer fee (2000won/3000won/4000won)
3. Admin()
    a. Transaction history

E. Console output (with REQ)

다음은 Term project assignment 의 Requirement 를 바탕으로 한 console output 이다.

==== Bank creation ==========

input: Kakao / 1 / Daegu / 2

```
--Start making Banks--
Insert the bank name: Kakao
[Bank Constructor] Bank name: Kakao

Do you want to create another bank?
1.Yes   2.No
1
Insert the bank name: Daegu
[Bank Constructor] Bank name: Daegu

Do you want to create another bank?
1.Yes   2.No
2
--Finish making Banks--
```

(REQ1.11) All accounts and ATMs shall be created and initialized during the program execution. : 모든 accounts 와 ATM 이 프로그램이 시작하고 난 후 initializing 된다.

==== **Account creation** ==========

input: Kakao / David / 111111111111 / 1004 / 20000 / 1
input: Daegu / Jane / 222222222222 / 2345 / 100000 / 1
input: Kakao / Kate / 333333333333 / 9876 / 150000 / 2

```
--Start making Accounts--
Insert the account's bank name : Kakao
Insert the account's owner name : David
Insert the account number : 111111111111
Insert the account password : 1004
Insert the account balance : 20000
[Account Constructor] Account ID: 111111111111

Bank Kakao's account list is updated as following.
num of accounts: 1
1. account number: 111111111111

Do you want to create another account?
1.Yes   2.No
1
Insert the account's bank name : Jane
Bank Jane does not exist.
Insert the account's bank name again : Daegu
Insert the account's owner name : Jane
Insert the account number : 222222222222
Insert the account password : 2435
Insert the account balance : 100000
[Account Constructor] Account ID: 222222222222

Bank Daegu's account list is updated as following.
num of accounts: 1
1. account number: 222222222222

Do you want to create another account?
1.Yes   2.No
1
Insert the account's bank name : Kakao
Insert the account's owner name : Kate
Insert the account number : 333333333333
Insert the account password : 9876
Insert the account balance : 150000
[Account Constructor] Account ID: 333333333333

Bank Kakao's account list is updated as following.
num of accounts: 2
1. account number: 111111111111
2. account number: 333333333333

Do you want to create another account?
1.Yes   2.No
2
--Finish making Accounts--
```

(REQ1.6) A user may have multiple Accounts in a Bank. : 사용자는 같은 은행의 여러 개의 계좌를 가질 수 있다. (이에 대한 출력은 밑에서 확인해볼 것이다)

input: Kakao / David / 444444444444 / 8282 / 400000

(REQ1.7) A user may have Accounts in multiple Banks. : 사용자는 여러 개의 은행에서 여러 개의 계좌를 가질 수 있다. (이에 대한 출력은 밑에서 확인해볼 것이다)

input: Daegu / David / 555555555555 / 2468 / 200000

==== ATM creation ==========

(REQ1.1) An ATM has a 6-digit serial number that can be uniquely identified among all ATMs (e.g., 315785). : ATM 은 각각마다 고유한 serial number 를 가진다.

(REQ1.2) An ATM is set to one of the following types: (1) Single Bank ATM, (2) Multi-Bank ATM. : ATM 은

single-bank, multi-bank 두 가지 type 을 가진다.

(REQ1.3) An ATM may support either unilingual or bilingual languages. : ATM 은 unilingual, 또는 bilingual 두 가지 방식을 지원한다

(REQ1.4) A Bank deposits a certain amount of cashes to an ATM to serve users. : ATM 은 unilingual, 또는 bilingual 두 가지 방식을 지원한다. ATM 은 동작을 위해 일정 금액을 예치하고 있어야한다.

input: Kakao / 111111 / 2 / 2 / 10 / 10 / 10 / 10 / 1
input: Daegu / 222222 / 1 / 1 / 10 / 10 /10 / 10 / 2

```
--Start making ATMs--
Insert the ATM's primary bank name : Kakao
Insert the ATM's serial number : 111111
Insert the ATM type :
1.Single Bank   2.Multi-Bank
2
Insert the ATM language type :
1.Unillingual   2.Bilingual
2
Insert the number of 1000 won cash in the ATM : 10
Insert the number of 5000 won cash in the ATM : 10
Insert the number of 10000 won cash in the ATM : 10
Insert the number of 50000 won cash in the ATM : 10
[ATM Constructor] ATM serial number: 111111
serial type: 2
serial lang: 2

Do you want to create another ATM?
1.Yes   2.No
1
Insert the ATM's primary bank name : Daegu
Insert the ATM's serial number : 222222
Insert the ATM type :
1.Single Bank   2.Multi-Bank
1
Insert the ATM language type :
1.Unillingual   2.Bilingual
1
Insert the number of 1000 won cash in the ATM : 10
Insert the number of 5000 won cash in the ATM : 10
Insert the number of 10000 won cash in the ATM : 10
Insert the number of 50000 won cash in the ATM : 10
[ATM Constructor] ATM serial number: 222222
serial type: 1
serial lang: 1

Do you want to create another ATM?
1.Yes   2.No
2
--Finish making ATMs--
```

==== ATM start ==========

1. **insert card**

(REQ1.5) A Bank can open an Account for user with necessary information to perform bank services. : Bank 는 user 의 service 이용을 위해서 account 를 열람할 수 있다.

(REQ8.1) An ATM that is configured with the bilingual support shall provide an option for a user to choose the preferred language either English or Korean. : ATM 이 bilingual 방식을 지원한다면 user 는 Ko 나 Eng 방식 중 한 가지를 선택해야 한다.

(REQ8.2) Once a certain language is chosen, all menus must be displayed using the chosen language. : 언어 선택이 완료되었다면 모든 menu 가 해당 언어로 보여지게 된다.

(REQ10.1) When a particular character (e.g. 'x') is given as a console input during the program execution, the following information shall be displayed to the console. (All ATMs' information: Remaining cash /All accounts' information: Remaining balance) : ATM 실행 도중 console input 으로 특정 값, "-1"이 입력되면 그 즉시 모든 ATM 의 보유 현금 정보와 모든 account 의 잔액 정보를 출력한다.

(REQ2.1) A session starts when a user inserts a card. : session 은 카드 입력 후 바로 시작된다.

(REQ3.3) An ATM shall ask a user to enter the password (e.g., Enter Password), and verify if the password is correct. : 계좌에 대한 비밀번호를 입력 받고 올바른 입력인지 확인한다.

(REQ3.4) If the entered password is incorrect, the ATM shall display an appropriate error message (e.g., Wrong Password). : 만약 비밀번호가 틀리면 에러 메시지를 출력한다.

(REQ3.5) If a user enters wrong passwords 3 times in a row, a session is aborted, and return the card to the user. : 세 번 이상 틀리면 session 을 종료시킨다.

input: 1 / 111111 / 1 / -1
input: 111111111111 / 8343 / 2342 / 8430

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어  2.English/영어 1
Your session will be started.
카드를 입력하시오 : -1

==== 모든 ATM, accounts의 정보 ====

ATM [고유 ATM 번호: 111111] ATM에 남아있는 잔액: 660000
(1000: 10장  , 5000: 10장  , 10000: 10장  , 50000: 10장 )
ATM [고유 ATM 번호: 222222] ATM에 남아있는 잔액: 660000
(1000: 10장  , 5000: 10장  , 10000: 10장  , 50000: 10장 )
계좌 [은행: Kakao, 계좌 번호: 111111111111, 계정주 : David] 계좌에 잔액: 20000

계좌 [은행: Kakao, 계좌 번호: 333333333333, 계정주 : Kate] 계좌에 잔액: 150000

계좌 [은행: Daegu, 계좌 번호: 222222222222, 계정주 : Jane] 계좌에 잔액: 100000

================================

카드를 입력하시오 : 111111111111
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
8342

[오류] 비밀번호가 틀렸습니다.
비밀번호를 입력해주세요.
2342

[오류] 비밀번호가 틀렸습니다.
비밀번호를 입력해주세요.
8430

[오류] 비밀번호가 틀렸습니다.
3회 연속으로 비밀번호가 틀려 세션이 종료됩니다.
인증 오류
세션이 종료됩니다.
```

*ATM(111111)*은 bilingual type ATM 이다. 이를 위해 처음에 언어 설정 관련 질문을 한다. 한국어를 입력하였으므로 한국어로 세션이 시작된다. 카드 번호 대신 -1 을 입력하면 ATM 의 보유 현금 정보와 모든 account 의 잔액 정보가 출력된다. *Account(111111111111)*의 비밀번호는 2435 로 설정 되어있다. 위 상황에서는 비밀번호를 3 회 이상 틀렸기 때문에 세션이 종료되는 것을 확인할 수 있다.

1. authorization

(REQ3.1) An ATM checks if the inserted card is valid for the current type of ATM. : authorization 절차에서는 카드가 ATM type 에 대해 유효한지 점검한다.

(REQ3.2) If an invalid card is inserted, the ATM shall display an appropriate error message (e.g., Invalid Card). : 만약 유효하지 않다면 error message 를 출력한다.

input: 1 / 222222 / 111111111111

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 222222
Your session will be started.
Insert card: 111111111111
Authorization Started
[ERROR] Invalid Card: the ATM is a single bank atm, and the card's bank is not the primary bank
Authorization failed
Session finished
```

*ATM(222222)*은 primary bank 가 Daegu 은행인 single bank type ATM 이다. 따라서 Kakao 은행의 account 인 *Account(11111111111)*가 입력되면 invalid card error 가 뜨고 세션이 종료된다.

```
Insert the serial number of the ATM you want to use : 222222
Your session will be started.
Insert card: 222222222222
Authorization Started
Insert password: 2435
```

*ATM(222222)*와 *Account(222222222222)*에 대해 새로운 session 을 시작한다.

==== Transactions - Deposit ==========

1. **Cash Deposit**

(REQ4.1) An ATM shall take either cash or check from a user. : ATM 은 cash 거래인지 check 거래인지 질문한다.

(REQ4.3) Once cash or checks are accepted by ATM, the transaction must be reflected to the bank account as well (i.e., the same amount of fund must be added to the corresponding bank account) : ATM 에서 일어나는 일들은 bank 와 account 에 반영되어야 한다.

(REQ1.8) Each ATM have several types of transaction fees, and paid as follows (Deposit fee for primary banks: KRW 0). : ATM 사용 과정에서는 수수료가 부과된다.

input: 1 / 1 / 3 / 0 / 0 / 0

```
================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 1

What are you depositing?1.cash deposit  2.check deposit 3. Exit
1

How much do you want to deposit?
Insert the amount of 1000won cash : 3

Insert the amount of 5000won cash : 0

Insert the amount of 10000won cash : 0

Insert the amount of 50000won cash : 0

Success! You have deposited 3000 won to your account.

The remaining balance in your Daegu account 222222222222 is 103000 won

The remaining 1000 in ATM : 13000
The remaining 5000 in ATM : 50000
The remaining 10000 in ATM : 100000
The remaining 50000 in ATM : 500000
```

*Account(222222222222)*는 초기에 100,000 원의 잔액을 가지고 있다. Cash Deposit 으로 1000 원권 3 장을 예금하였으므로 계좌의 잔액은 103,000 원으로 바뀌게 된다. 해당 거래는 primary bank 에 대한 deposit 이므로 수수료가 부과되지 않는다. *ATM(222222)*보유한 1000 원권 금액이 3000 원 증가한다.

(REQ4.2) An ATM shall display an appropriate error message if the number of the inserted cash or checks exceed the limit allowed by the ATM. (Cash Deposit limit is about 30 paper cashes): 일정 금액 이상의 입금에 대해서는 입금 오류를 출력한다.

input: 1 / 1/ 35 / 0 / 0 / 0

```
================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 1

What are you depositing?1.cash deposit  2.check deposit 3. Exit
1

How much do you want to deposit?
Insert the amount of 1000won cash : 35

Insert the amount of 5000won cash : 0

Insert the amount of 10000won cash : 0

Insert the amount of 50000won cash : 5

[ERROR] Your deposit amount is over 30(cash deposit limit).
```

(REQ2.3) When a session ends, the summary of all transactions performed in a session must be displayed. : Session 이 종료되면 거래에 대한 요약을 출력시킨다.

input: 4

```
==============================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 4

Transaction ID: 0| Cash Deposit | Bank: Daegu| Account Number: 222222222222 / Deposit Amount: 3000
Session finished
```

2. Check Deposit

(REQ4.6) The deposited check does not increase available cash in ATM that can be used by other users. : 수표 사용에 따른 ATM 내의 사용 금액 변화는 없다.

(REQ1.8) Each ATM have several types of transaction fees, and paid as follows: (Deposit fee for non-primary banks: KRW 1,000; the fee is paid by inserting additional cash) : non-primary bank에 대한 거래 수수료는 1000원이다.

input: 1 / 222222222222 / 2345 / 1 / 2 / 2000 / 30000 / 1

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어  2.English/영어 1
Your session will be started.
카드를 입력하시오 : 222222222222
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
2435

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체   4.취소 : 1

어떤 형태로 예금을 하십니까?1.화폐       2.수표          3. 취소
2

예금하고자 수표의 장 수를 입력해주세요.
2

1번째 수표의 금액을 입력해주세요. : 2000
2번째 수표의 금액을 입력해주세요. : 30000

예금 수수료로 1000원이 청구됩니다.
1.추가 현금 넣기         2.취소          : 1
32000원이 계좌로 입금되었습니다.

Daegu 은행의 222222222222계좌에 남아있는 총 금액은 135000원입니다.

ATM에 존재하는 천원권의 총 금액 : 11000
ATM에 존재하는 오천원의 총 금액 : 50000
ATM에 존재하는 만원권의 총 금액 : 100000
ATM에 존재하는 오만원의 총 금액 : 500000

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체   4.취소 : 4

거래 고유 번호: 0| 수표 예금 | 은행: Daegu| 계좌번호 : 222222222222 / 예금 금액: 32000
세션이 종료됩니다.
```

Check deposit 과정이다. Kakao 은행을 primary bank 로 가지는 Multi bank *ATM(111111)*을 통해서 Daegu 은행의 *Account(222222222222)*로의 check deposit 을 실행한다. 이 때는 bank 가 서로 다르므로 1000 원의 수수료가 부과된다.

(REQ10.1) When a particular character (e.g. 'x') is given as a console input during the program execution, the following information shall be displayed to the console. : 특정 문자가 입력되면 존재하는 모든 ATM 과 Account 의 정보를 출력한다.

(REQ4.5) The deposited cash increase available cash in ATM that can be used by other users. : 입금된 금액 만큼 ATM 의 사용가능 금액도 증가한다.

input: -1

```
Insert card: -1

==== all ATM, accounts info ====

ATM [SN: 111111] remaining cash: 661000(1000: 11, 5000: 10, 10000: 10, 50000: 10)
ATM [SN: 222222] remaining cash: 663000(1000: 13, 5000: 10, 10000: 10, 50000: 10)
Account [Bank: Kakao, No: 111111111111, Owner: David] balance: 20000

Account [Bank: Kakao, No: 333333333333, Owner: Kate] balance: 150000

Account [Bank: Daegu, No: 222222222222, Owner: Jane] balance: 135000

==============================
```

-1이 입력되면 ATM의 보유 현금 정보와 모든 account의 잔액 정보가 출력된다. ATM의 보유 현금이 업데이트된 것을 확인할 수 있다.

==== Transactions - Withdrawal ==========

(REQ5.1) An ATM shall ask a user to enter the amount of fund to withdraw. : withdrawal 과정이 시작되면 사용자에게 출금 금액을 물어 보아야한다.

(REQ5.3) Once the withdrawal is successful, the transaction must be reflected to the bank account as well (i.e., the same amount of fund must be deducted from the corresponding bank account). : ATM에서의 transection은 Account와 bank에서도 반영 되어야한다.

(REQ5.4) Some withdrawal fee may be charged (See REQ in System Setup). : 수수료가 부과된다.

(REQ5.5) The cash withdrawal lower available cash in the ATM that can be used by other users. : 인출이 되면 ATM의 사용 가능 금액은 감소한다.

(REQ5.6) The maximum number of withdrawals per each session is 3.- If a user wants to withdraw four times, it needs to end the current session after withdrawing three times and restart another session for one more withdrawal. : 한 session당 출금 횟수는 최대 3회이다.

input: 222222 / 222222222222 / 2345 / 2 / 5000 / 15000 / 2 / 1000 / 3000 / 2 / 10000 / 20000 / 2

```
Insert the serial number of the ATM you want to use : 222222
Your session will be started.
Insert card: 222222222222
Authorization Started
Insert password: 2435


================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 2

Insert the amount of cash you want to withdrawal.
What denomination of bills would you like to withdraw? (1,000 won, 5,000 won, 10,000 won, 50,000 won)
please input 1000, 5000, 10000, 50000...: 5000

How many bills of would you like to withdraw?
please input 5000, 10000, 15000...: 15000

You are charged for withdrawal fee of 1000won.
Success!

The remaining balance in your Daegu account is 119000 won

The remaining 1000 in ATM : 13000
The remaining 5000 in ATM : 35000
The remaining 10000 in ATM : 100000
The remaining 50000 in ATM : 500000


================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 2

Insert the amount of cash you want to withdrawal.
What denomination of bills would you like to withdraw? (1,000 won, 5,000 won, 10,000 won, 50,000 won)
please input 1000, 5000, 10000, 50000...: 1000

How many bills of would you like to withdraw?
please input 1000, 2000, 3000...: 3000

You are charged for withdrawal fee of 1000won.
Success!

The remaining balance in your Daegu account is 115000 won

The remaining 1000 in ATM : 10000
The remaining 5000 in ATM : 35000
The remaining 10000 in ATM : 100000
The remaining 50000 in ATM : 500000
```

```
================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 2

Insert the amount of cash you want to withdrawal.
What denomination of bills would you like to withdraw? (1,000 won, 5,000 won, 10,000 won, 50,000 won)
please input 1000, 5000, 10000, 50000...: 10000

How many bills of would you like to withdraw?
please input 10000, 20000, 30000...: 20000

You are charged for withdrawal fee of 1000won.
Success!

The remaining balance in your Daegu account is 94000 won

The remaining 1000 in ATM : 10000
The remaining 5000 in ATM : 35000
The remaining 10000 in ATM : 80000
The remaining 50000 in ATM : 500000


================================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 2

[Error] If you want more withdrawal, Please restart session
```

*ATM(222222)*에서 출금이 진행되었다. 위를 실행하면 *Account(222222222222)*로부터 5000 원권 3 장, 1000 원권 3 장, 10000 원권 2 장이 빠져나가게 된다. (총 38000 원 출금)

해당 Withdrawal 과정은 ATM primary bank에 대한 활동이므로 한 번의 인출 과정마다 1000원의 수수료가 부과된다. 출금을 3번 이상의 출금을 진행하려고 하면, 경고 문구가 뜨고, 추가 출금 과정이 불가능하도록 설계하였다.

input: -1 / 4

```
==============================
Choose transaction

1.Deposit        2.Withdrawal   3.Transfer      4.Exit : -1

==== all ATM, accounts info ====

ATM [SN: 111111] remaining cash: 661000(1000: 11, 5000: 10, 10000: 10, 50000: 10)
ATM [SN: 222222] remaining cash: 625000(1000: 10, 5000: 7, 10000: 8, 50000: 10)
Account [Bank: Kakao, No: 111111111111, Owner: David] balance: 20000

Account [Bank: Kakao, No: 333333333333, Owner: Kate] balance: 150000

Account [Bank: Daegu, No: 222222222222, Owner: Jane] balance: 94000

==============================

Choose transaction
1.Deposit        2.Withdrawal   3.Transfer      4.Exit : 4

Transaction ID: 1| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 15000
Transaction ID: 2| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 3000
Transaction ID: 3| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 20000
Session finished
```

-1을 입력하면 ATM과 Account의 정보가 출력된다.

*Account(222222222222)*의 기존 잔액은 135000원이었다. 38000원을 출금한 이후에 (수수료 3000원) 94000원으로 잔액이 바뀐 것을 확인할 수 있다.

ATM(222222)의 보유 현금도 다음과 같이 바뀐 것을 확인할 수 있다. (기존의에는 1000원:13장, 5000원:10장, 10000원:10장, 50000원:10장이 ATM에 예치 되어있었다)

Withdrawal 실행 이후 세션을 종료하면 3회의 인출에 대한 transaction history 내역이 출력된다.

(REQ5.2) An ATM shall display an appropriate error message if there is insufficient fund in the account or insufficient cash in the ATM. : ATM이나 계좌에 충분한 돈이 있지 않을 경우 error를 출력시킨다.

input: 111111 / 1 / 222222222222 / 2345 / 2 / 1000 / 600000 / 2 / 5000 / 500000

```
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어 2.English/영어1
Your session will be started.
카드를 입력하시오 : 222222222222
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
2435

진행하실 거래를 선택해주세요.
1.예금   2.출금          3.이체   4.취소 : 2

인출하고자 하는 금액을 적으시오.
인출하고자 하는 지폐의 종류를 적으시오. (1,000 원, 5,000원, 10,000원, 50,000원)
[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: 1000

인출하고자 하는 천원권의 총 금액을 입력하시오.
[주의] 1000, 2000, 3000 단위로 적으시오. 600000

[오류] ATM에 들어있는 돈이 부족합니다.

진행하실 거래를 선택해주세요.
1.예금   2.출금          3.이체   4.취소 : 2

인출하고자 하는 금액을 적으시오.
인출하고자 하는 지폐의 종류를 적으시오. (1,000 원, 5,000원, 10,000원, 50,000원)
[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: 50000

인출하고자 하는 오만원권의 총 금액을 입력하시오.
[주의] 50000, 100000, 150000 단위로 적으시오. 500000

[오류] 계좌에 충분한 금액이 들어있지 않습니다.
```

(REQ5.7) The maximum amount of cash withdrawal per transaction is KRW 500,000. : Withdrawal 의 최대 금액은 500,000 원이다

input: 1 / 111111 / 1 / 333333333333 / 9876 / 2 / 50000 / 1000000

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어 2.English/영어1
Your session will be started.
카드를 입력하시오 : 333333333333
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
9876

진행하실 거래를 선택해주세요.
1.예금   2.출금          3.이체   4.취소 : 2

인출하고자 하는 금액을 적으시오.
인출하고자 하는 지폐의 종류를 적으시오. (1,000 원, 5,000원, 10,000원, 50,000원)
[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: 50000

인출하고자 하는 오만원권의 총 금액을 입력하시오.
[주의] 50000, 100000, 150000 단위로 적으시오. 1000000

[오류] 오십만원이 넘는 인출은 불가능합니다.
```

==== Transactions – Transfer ==========

(REQ6.1) An ATM shall ask a user to choose the transfer types either cash transfer or account fund transfer. : transfer 과정에서는 사용자에게 현금 이체를 원하는지 계좌 이체를 원하는지 물어보아야 한다.

(REQ6.2) For both cash and account transfers, an ATM shall ask the destination account number where the fund is to be transferred. : 이체할 계좌(destination account)를 물어보아야 한다.

(REQ6.5) Some transfer fee may be charged (See REQ in System Setup). : 수수료가 존재한다.

(REQ6.6) The inserted cash for transfer increase available cash in ATM that can be used by other users. : 현금이체

이후에는 ATM의 사용가능 금액이 늘어나게 된다.

(REQ6.7) Once the transfer is successful, the transaction must be reflected to the bank account as well (i.e., the same amount of fund must be deducted from the source bank account, and then added to the destination bank account). :
모든 과정은 bank와 account에 반영되어야 한다.

(REQ1.8) Each ATM have several types of transaction fees, and paid as follows: (Account transfer fee between primary banks: KRW 2,000; the fee is paid from the source account, Account transfer fee between primary and non-primary banks: KRW 3,000; the fee is paid from the source account, Account transfer fee between non-primary banks: KRW 4,000; the fee is paid from the source account, Cash transfer fee to any bank type: KRW 5,000; the fee is paid by inserting additional cash. : 수수료는 위와 같이 부과된다.

1. Cash Transfer

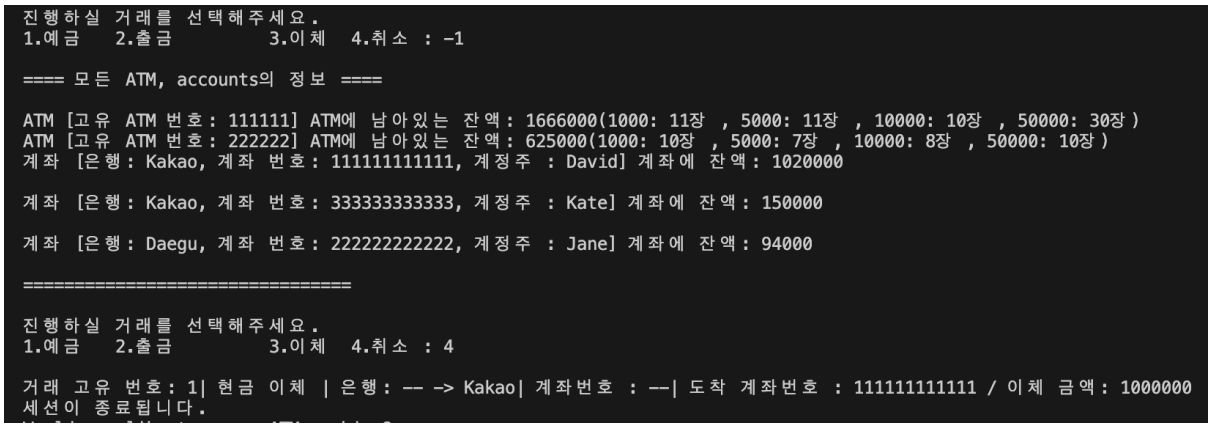
(REQ6.3) For cash transfer, an ATM shall ask user to insert the cash including the transaction fees, and verify if the amount of the inserted cash is correct. All inserted cash excluding the transaction fee shall be transferred. :
cash transfer 과정에서의 수수료는 insert cash에 포함되어야 한다.
input: 1 / 111111 / 1 / 333333333333 / 9876 / 3 / 1 / 111111111111 / 0 / 1 / 0 / 20 / Y

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어  2.English/영어1
Your session will be started.
카드를 입력하시오 : 333333333333
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
9876

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체  4.취소 : 3

이체 종류를 고르시오. 1.현금 이체       2.계좌 이체     3. 취소 1

돈을 보내려는 계좌를 고르시오. 111111111111

이체하고자 하는 금액을 적으시오.
( 이체 수수료 5,000원이 입금된 금액으로부터 제외될 것입니다. )
1,000원 지폐 개수 : 0

5,000원 지폐 개수 : 1

10,000원 지폐 개수 : 0

50,000원 지폐 개수 : 20

총 이체 금액은 1000000 원 입니다. (동의할 경우, Y를 입력해주세요.) : Y

성공적으로 1000000원을 다음 도착 계좌로 이체하였습니다 -> 111111111111
현금 이체 결과로,
ATM에 존재하는 천원권의 총 금액 : 11000
ATM에 존재하는 오천원의 총 금액 : 55000
ATM에 존재하는 만원권의 총 금액 : 100000
ATM에 존재하는 오만원의 총 금액 : 1500000
111111111111 계좌 잔액은  1020000 원 입니다.
이체 성공.
```

이체 과정은 *ATM(111111)*에서 진행된다. *Account(333333333333)*에서 *Account(111111111111)*로 현금 이체가 진행된다. ATM에 1005,000원이체를 위해 현금 투입

과정이 진행되고, 현금 이체 수수료 5000원을 제외한 1,000,000원이 *Account(11111111111)*로 입금된다.

input: -1



-1을 입력해서 결과를 확인해보면 기존 20,000원을 가지고 있던 *Account(11111111111)*의 잔액이 1,020,000으로 바뀐 것을 확인할 수 있다. 마찬가지로 *ATM(111111)*에도 50,000권이 20장 수수료로써 부과된 5,000원권이 1장 추가된 것을 확인할 수 있다.

Cash transfer 이후 세션을 종료하면 현금 이체 transaction history가 출력된다.

2. Account Transfer

(REQ6.4) For account transfer, an ATM shall ask the source account number, and the amount of fund to be transferred. : 계좌 이체 과정에서는 돈을 뽑으려는 계좌와 보내는 금액을 확인해야한다.

input: 111111 / 1 / 111111111111 / 1004 / 3 / 2 / 3333333333333 / 1 / 900000 / -1

```
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어 2.English/영어1
Your session will be started.
카드를 입력하시오 : 111111111111
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
1004

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체   4.취소 : 3

이체 종류를 고르시오. 1.현금 이체       2.계좌 이체     3. 취소 2

돈을 보내려는 계좌를 고르시오. 333333333333

돈을 빼려는 계좌가 111111111111가 맞습니까?
숫자를 입력해주세요 -> 1.네     2.아니요 : 1

이체를 위한 금액을 적어주세요.
900000

수수료 2000 원이 해당 계좌에서 인출됩니다.
성공적으로 900000원을 다음 도착 계좌로 이체하였습니다 -> 333333333333
계좌 이체 결과로,
출발 계좌에 존재하는 금액은 Kakao 계좌, 111111111111 에서 118000원입니다.
도착 계좌에 존재하는 금액은 Kakao 계좌, 333333333333 에서 1050000원입니다.
이체 성공.
진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체   4.취소 : -1

==== 모든 ATM, accounts의 정보 ====

ATM [고유 ATM 번호: 111111] ATM에 남아있는 잔액: 1666000(1000: 11장 , 5000: 11장 , 10000: 10장 , 50000: 30장)
ATM [고유 ATM 번호: 222222] ATM에 남아있는 잔액: 625000(1000: 10장 , 5000: 7장 , 10000: 8장 , 50000: 10장)
계좌 [은행: Kakao, 계좌 번호: 111111111111, 계정주 : David] 계좌에 잔액: 118000

계좌 [은행: Kakao, 계좌 번호: 333333333333, 계정주 : Kate] 계좌에 잔액: 1050000

계좌 [은행: Daegu, 계좌 번호: 222222222222, 계정주 : Jane] 계좌에 잔액: 94000

==============================
```

계좌이체는 *ATM(111111)*에서 진행된다. *Account(11111111111)*에서 *Account(333333333333)*로 계좌이체를 진행하였다. 총 900,000원이 이체되었고, 2000원(primary to primary)이 수수료로써 빠져나간다.

-1을 입력해 결과를 확인해보면 902,000원이 *Account(11111111111)*에서 빠져나간 것을 확인할 수 있다.
*Account(333333333333)*에는 900,000이 추가 되어있다. 계좌이체 과정이므로 ATM의 잔액은 바뀌지 않는다.

==== Display of Transaction History (Admin Menu) ==========

(REQ7.1) When a session is started by an admin by inserting an admin card (See REQ in System Setup), an ATM displays a menu of "Transaction History" only. : admin카드가 들어오면 display of transaction history가 시작된다. 이 때 menu는 Transaction History가 유일하다.

(REQ7.2) When the "Transaction History" menu is selected, an ATM display the information of all transactions from all users from the beginning of the system start. : Transaction History menu를 누르면 system이 시작한 이후에 진행된 모든 transaction의 결과를 출력시켜야한다.

(REQ7.3) The "Transaction History" information shall be outputted to the external file (e.g., txt file). : Transaction History 는 외부 파일로 출력된다.

Session이 시작될 때, “Insert card:” 입력 창에 계좌 번호가 아닌, Admin card로 “000000” 번호를 입력하면 “1. Transaction History”가 console 화면에 뜬다. 이 때 1을 입력하면 해당 ATM 기기의 전체 Transaction History가 출력된다.

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어  2.English/영어2
Your session will be started.
Insert card: 000000
Please select your transaction
1. Transaction History:
1
Transaction ID: 0| Check Deposit | Bank: Daegu| Account Number: 222222222222 / Deposit Amount: 32000
Transaction ID: 1| Cash Transfer | Bank: -- -> Kakao| Account Number: --| Destination Account Number: 111111111111 / Transfer Amount: 1000000
Transaction ID: 2| Account Transfer | Bank: Kakao -> Kakao| Account Number: 111111111111| Destination Account Number: 333333333333 / Transfer Amount: 900000
```

*ATM(111111)*의 전체 Transaction History이다.

ATM_Transaction.txt

```
1   Transaction ID: 0| Check Deposit | Bank: Daegu| Account Number: 222222222222 / Deposit Amount: 32000
2   Transaction ID: 1| Cash Transfer | Bank: -- -> Kakao| Account Number: --| Destination Account Number: 111111111111 / Transfer Amount: 1000000
3   Transaction ID: 2| Account Transfer | Bank: Kakao -> Kakao| Account Number: 111111111111| Destination Account Number: 333333333333 / Transfer Amount: 900000
4
5
```

*ATM(111111)*의 ATM_Transaction.txt 문서이다.

```
Insert the serial number of the ATM you want to use : 222222
Your session will be started.
Insert card: 000000
Please select your transaction
1. Transaction History:
1
Transaction ID: 0| Cash Deposit | Bank: Daegu| Account Number: 222222222222 / Deposit Amount: 3000
Transaction ID: 1| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 15000
Transaction ID: 2| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 3000
Transaction ID: 3| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 20000
```

*ATM(222222)*의 전체 Transaction History이다.

ATM_Transaction.txt

```
1   Transaction ID: 0| Cash Deposit | Bank: Daegu| Account Number: 222222222222 / Deposit Amount: 3000
2   Transaction ID: 1| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 15000
3   Transaction ID: 2| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 3000
4   Transaction ID: 3| Withdrawal | Bank: Daegu| Account Number: 222222222222 / Withdrawal Amount: 20000
5
6
```

*ATM(222222)*의 ATM_Transaction.txt 문서이다.

==== Exceptional Handling ==========

(REQ9.1) An ATM shall display an appropriate message for each exception scenario (both explicitly stated in this document and implicitly assumed ones), and take an appropriate action (e.g., print an appropriate error message, end a session). : ATM이 가지고 있는 돈이 부족한 경우, 인출을 4번 이상 시도한 경우 등등 Session이 끝나거나, 이에 적합한 Error Message를 출력한다.

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어 2.English/영어1
Your session will be started.
카드를 입력하시오. : 111111111111
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
1004

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체  4.취소 : 2

인출하고자 하는 금액을 적으시오.
인출하고자 하는 지폐의 종류를 적으시오. (1,000 원, 5,000원, 10,000원, 50,000원)
[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: 1000

인출하고자 하는 천원의 갯수를 고르시오
[주의] 1000, 2000, 3000 단위로 적으시오. 20000

[오류] ATM에 들어있는 돈이 부족합니다.
```

ATM에 1000원 지폐 10장이 존재하는데, 총 20장을 인출하려고 시도할 경우 ATM에 가지고 있는 현금 지폐 수를 초과하기 때문에 “들어있는 돈이 부족하다” 는 에러 메시지를 보낸다.

```
ATM에 남아있는 천원의 갯수 : 8
ATM에 남아있는 오천원의 갯수 : 9
The remaining 10000 in ATM : 10
The remaining 50000 in ATM : 10

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체  4.취소 : 2

인출하고자 하는 금액을 적으시오.
인출하고자 하는 지폐의 종류를 적으시오. (1,000 원, 5,000원, 10,000원, 50,000원)
[주의] 1000, 5000, 10000, 50000 단위로 적으시오.: 10000

인출하고자 하는 만원의 갯수를 고르시오
[주의] 10000, 20000, 30000 단위로 적으시오. 10000

인출 수수료로 1000원이 청구됩니다.
성공하였습니다.

Kakao의 계좌는 0원이 되었습니다.

ATM에 남아있는 천원의 갯수 : 8
ATM에 남아있는 오천원의 갯수 : 9
The remaining 10000 in ATM : 9
The remaining 50000 in ATM : 10

진행하실 거래를 선택해주세요.
1.예금   2.출금         3.이체  4.취소 : 2

[오류] 추가적으로 인출하려면 새로운 세션을 진행하세요.
```

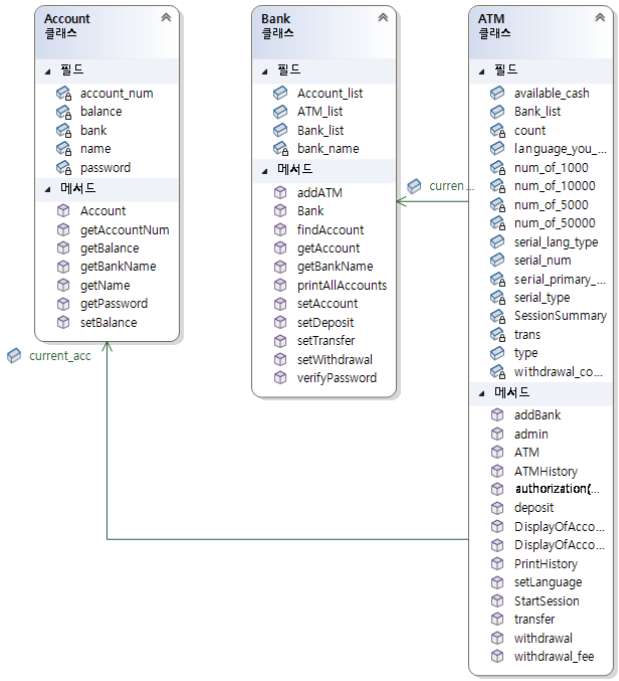
단일 Session 내에서 인출을 3번 이상 시도할 경우, 추가적인 인출을 위해서는 새로운 세션이 진행되어야 하기 때문에, 오류 메시지를 출력한다.

```
Would you like to use a ATM machine?
1.Yes   2.No
1
Insert the serial number of the ATM you want to use : 111111
Which language do you want to use? 어떤 언어를 사용하고 싶으신가요?
1.Korean/한국어  2.English/영어1
Your session will be started.
카드를 입력하시오 : -1

==== 모든 ATM, accounts의 정보 ====

ATM [고유 ATM 번호: 111111] ATM에 남아있는 잔액: 660000
(1000: 10장  , 5000: 10장  , 10000: 10장  , 50000: 10장 )
ATM [고유 ATM 번호: 222222] ATM에 남아있는 잔액: 660000
(1000: 10장  , 5000: 10장  , 10000: 10장  , 50000: 10장 )
계좌 [은행: Kakao, 계좌 번호: 111111111111, 계정주 : David] 계좌에 잔액: 20000

계좌 [은행: Kakao, 계좌 번호: 333333333333, 계정주 : Kate] 계좌에 잔액: 150000

계좌 [은행: Daegu, 계좌 번호: 222222222222, 계정주 : Jane] 계좌에 잔액: 100000

==============================

카드를 입력하시오 : 111111111111
인증 절차를 시작합니다.
비밀번호를 입력해주세요.
8342

[오류] 비밀번호가 틀렸습니다.
비밀번호를 입력해주세요.
2342

[오류] 비밀번호가 틀렸습니다.
비밀번호를 입력해주세요.
8430

[오류] 비밀번호가 틀렸습니다.
3회 연속으로 비밀번호가 틀려 세션이 종료됩니다.
인증 오류
세션이 종료됩니다.
```

위에서 언급했듯이 ATM 내에서 추가로 Password를 3회 연속으로 틀린 경우도 이에 관한 오류 메시지를 사용자에게 보여주고, 인증 오류의 사유로 세션이 종료됨을 알린다.

F. class diagram, concepts of object-oriented programming

다음은 class diagram 이다. Visual studio 를 활용해서 그려보았다.



다음은 본 프로젝트에서 사용한 OOP(object-oriented programming) 개념이다.

a. Abstraction

1. Bank, Account, ATM에 관한 정보들을 입력할 때 사용자는 원리, 설계를 모르더라도 정보만 입력한다면 이 정보에 적한 은행, 계좌, ATM기를 만들 수 있다.

2. ATM에서 Deposit, Withdraw, Transfer 등 사용자가 거래를 원할 때, 내부 구현을 몰라도 원하는 금액, 거래를 입력만 하더라도 사용할 수 있다.

b. Encapsulation

1. 각각의 class는 private을 통해 외부에서 중요 변수들에 바로 접근할 수 없고, public에서 선언한 호출 함수들로 private 변수들에 접근할 수 있다.

2. ATM 내부의 거래 내역들을 담을 수 있는 SessionSummary를 private으로 보호하여 외부에서 접근하는 것이 아니라 오로지 admin을 통해서만 접근할 수 있다.

c. STL

1. STL의 Vector를 활용해 bank_list, atm_list를 dynamic array로 만들어 이중 포인터를 사용했을 때보다 메모리의 수동적인 할당이 가능해 효율적이다.

G. Member student contribution

All students equally contributed