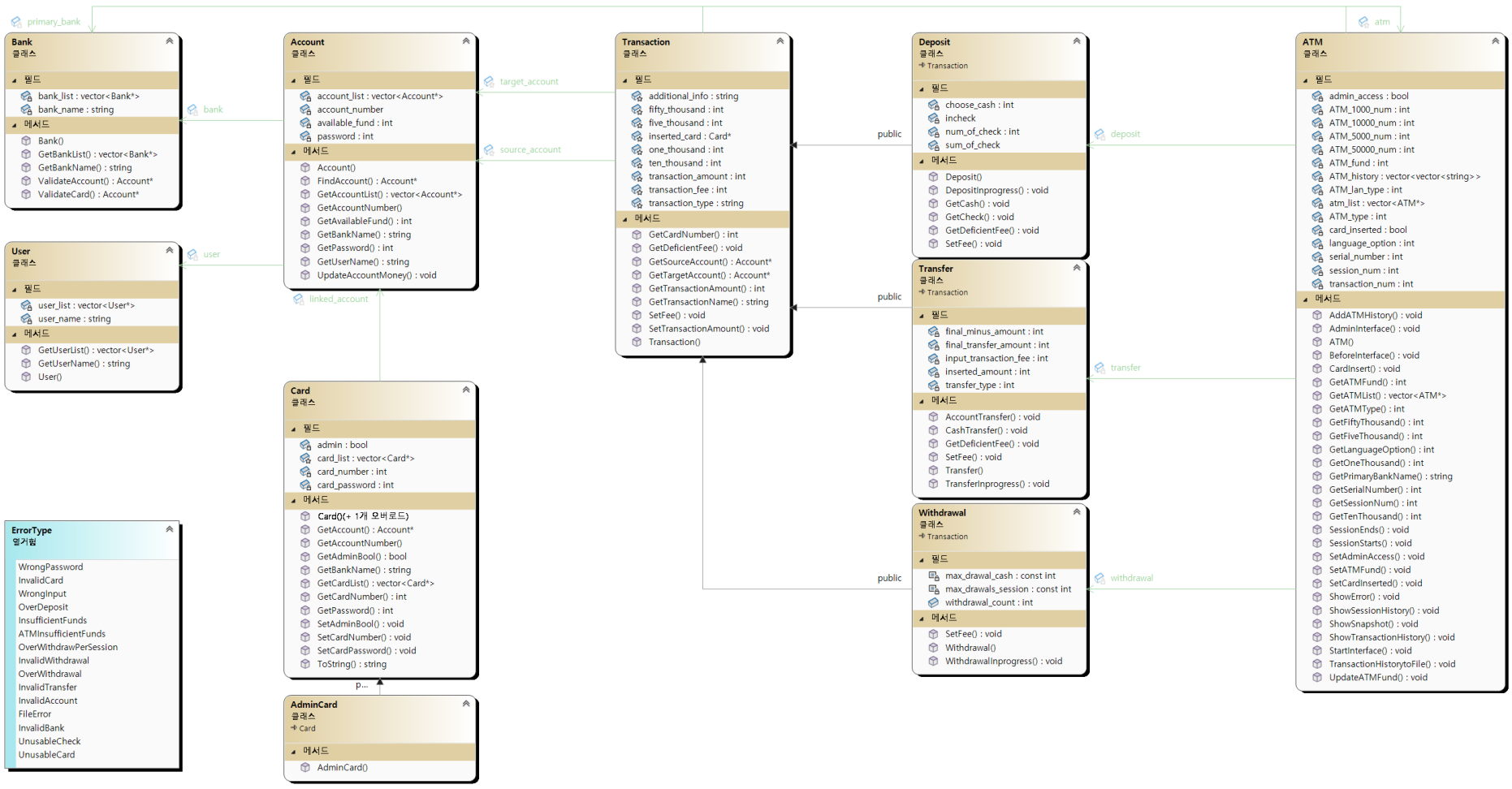
Term Project Final Report

1. Team Members

- 202311053 김예진
- 202311055 김유진
- 202311083 박은빈
- 202311104 안서연

2. Final Class Diagram Design



3. Implemented Requirements (Requirement Check Sheet)

Requirement Check Sheet

1. System Setup							
1.1	O	1.2	O	1.3	O	1.4	O
1.5	O	1.6	O	1.7	O	1.8	O
1.9	O	1.10	O	1.11	O		
2. ATM Session							
2.1	O	2.2	O	2.3	O		
3. User Authorization							
3.1	O	3.2	O	3.3	O	3.4	O
3.5	O						
4. Deposit							
4.1	O	4.2	O	4.3	O	4.4	O
4.5	O	4.6	O				
5. Withdrawal							
5.1	O	5.2	O	5.3	O	5.4	O
5.5	O	5.6	O	5.7	O		
6. Transfer							
6.1	O	6.2	O	6.3	O	6.4	O
6.5	O	6.6	O	6.7	O		
7. Display of Transaction History (Admin Menu)							
7.1	O	7.2	O	7.3	O		
8. Multi-language support							
8.1	O	8.2	O				
9. Exception Handling							
9.1	O						
10. Display of Account/ATM Snapshot							
10.1	O						

4. Implemented Requirements (Console Screenshots)

0. Test Case Initialization

REQ 1~10이 제대로 구현됨을 보이기 위해 코드를 실행시킬 때, 정보를 다음과 같이 초기화하여 사용하였다. 1~10까지는 ATM과 Account 번호만 캡션에 넣어 설명하였다.

▼ Banks

NH, Hana, 우리, Daegu

▼ Users

은빈, 예진, 유진, 서연

▼ Accounts

- Account 1
  - Bank: NH
  - User: 은빈
  - Account number: 110011001100
  - Available funds: 100000[십만원]
  - Card password: 1100
  - Card number: 11001100
- Account 2
  - Bank: NH
  - User: 예진
  - Account number: 220022002200
  - Available funds: 5000
  - Card password: 2200
  - Card number: 22002200
- Account 3
  - Bank: Hana
  - User: 유진
  - Account number: 330033003300
  - Available funds: 300000[삼십만원]
  - Card password: 3300
  - Card number: 33003300
- Account 4
  - Bank: 우리
  - User: 서연

    - Account number: 440044004400
    - Available funds: 30000
    - Card password: 4400
    - Card number: 44004400
- Account 5
    - Bank: Daegu
    - User: 예진
    - Account number: 550055005500
    - Available funds: 1000000(백만원)
    - Card password: 5500
    - Card number: 55005500
- Account 6
    - Bank: Hana
    - User: 유진
    - Account number: 660066006600
    - Available funds: 1000
    - Card password: 6600

▼ ATMs

- ATM 1
    - Primary: NH
    - Serial: 111111
    - Type: 1
    - Language: 1
    - Initial Funds: 50, 2, 1, 0

      (50000원, 10000원, 5000원, 1000원 지폐 개수)
- ATM 2
    - Primary: NH
    - Serial: 222222
    - Type: 2
    - Language: 2
    - Initial Funds: 0, 3, 1, 2
- ATM 3
    - Primary: Hana
    - Serial: 333333
    - Type: 2
    - Language: 1
    - Initial Funds: 0, 0, 1, 50
- ATM 4
    - Primary: 우리
    - Serial: 444444
    - Type: 1
    - Language: 2
    - Initial Funds: 5, 5, 5, 5
- ATM 5
    - Primary: Daegu
    - Serial: 555555
    - Type: 2
    - Language: 2
    - Initial Funds: 2, 3, 4, 5

1. System Setup

▼ (REQ1.1) An ATM has a 6-digit serial number that can be uniquely identified among all ATMs (e.g., 315785).

```
class ATM {
private:
    int serial_number;          // 6-digit
    int ATM_type;          // single = 1, multi = 2
    Bank* primary_bank;
    int ATM_lan_type;          // unilingual = 1, bilingual = 2
    static int language_option;          // English = 1, Korean = 2
    int ATM_fund;
    int ATM_1000_num = 0, ATM_5000_num = 0, ATM_10000_num = 0, ATM_50000_num = 0;
    static int session_num;
    static int transaction_num;
    int transaction_count = 0;         // transaction number in one session
    bool card_inserted = false;
    bool admin_access = false;
    static vector<vector<string>> ATM_history;
    static vector<ATM*> atm_list;
    Deposit* deposit;
    Transfer* transfer;
    Withdrawal* withdrawal;
```

[REQ1.1.1] class ATM의 private variable 6-digit serial number

- ATM class에서 private member로 int 형태의 serial number를 만들어 각 ATM이 unique serial number을 가질 수 있도록 구현하였다.
- int serial_number을 통해 ATM serial number은 6자리 정수로 받도록 설정했다.

```
================================ ATM ================================
Please select the ATM to use.
Available ATMs:
1: ATM [SN: 111111, Bank: NH, Type: 1]
2: ATM [SN: 222222, Bank: NH, Type: 1]
3: ATM [SN: 333333, Bank: Hana, Type: 2]
4: ATM [SN: 444444, Bank: 우리, Type: 1]
5: ATM [SN: 555555, Bank: Daegu, Type: 2]

0: Return to Main Menu
```

[REQ1.1.2] ATM의 고유한 SN

- 각각의 ATM에 6자리의 고유한 serial number가 할당되어 있음을 알 수 있다.

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

```
class ATM {
private:
    int serial_number;          // 6-digit
    int ATM_type;          // single = 1, multi = 2
    Bank* primary_bank;
    int ATM_lan_type;          // unilingual = 1, bilingual = 2
    static int language_option;          // English = 1, Korean = 2
    int ATM_fund;
    int ATM_1000_num = 0, ATM_5000_num = 0, ATM_10000_num = 0, ATM_50000_num = 0;
    static int session_num;
    static int transaction_num;
    bool card_inserted = false;
    bool admin_access = false;
    vector<vector<string>> ATM_history;
    static vector<ATM*> atm_list;
    Deposit* deposit;
    Transfer* transfer;
    Withdrawal* withdrawal;
```

[REQ1.2.1] class ATM의 private variable ATM_type

- int ATM_type을 통해 single bank는 정수 1, multi bank는 정수 2로 표시하도록 했다.

```
================================== ATM ==================================
Please select the ATM to use.
Available ATMs:
1: ATM [SN: 111111, Bank: NH, Type: 1]
2: ATM [SN: 222222, Bank: NH, Type: 2]
3: ATM [SN: 333333, Bank: Hana, Type: 2]
4: ATM [SN: 444444, Bank: 우리, Type: 1]
5: ATM [SN: 555555, Bank: Daegu, Type: 2]

0: Return to Main Menu
=========================================================================
1
Selected ATM [Bank: NH, SN: 111111]
This ATM only suppports English.
=========================================================================
Please insert your card.
33003300
==================================error==================================
Error: Unusable Card for this ATM.
Returning to main ATM menu.
================================== ATM ==================================
```

[REQ1.2.2] ATM_type = 1일 때 다른 은행 카드를 넣어 error 발생

- single bank를 지원하는 ATM의 경우, 다른 은행 카드를 넣으면 작동하지 않는 모습을 볼 수 있다. 사용된 ATM 1의 주거래 은행은 NH, card의 은행은 Hana이다.

▼ (REQ1.3) An ATM may support either unilingual or bilingual languages.

1.3(1) When an ATM is configured unilingual, all information is displayed in English only.

```
=========================================================================
1
Selected ATM [Bank: NH, SN: 111111]
This ATM only suppports English.
=========================================================================
Please insert your card.
11001100
=========================================================================
Enter the card password.
=========================================================================
1100
=========================================================================
Card successfully inserted!
=========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
=========================================================================
```

[REQ1.3.1] Unilingual ATM 1 선택 시 화면

- unilingual ATM을 선택하면, ‘This ATM only supports English.’라는 메시지가 출력되면서 모든 정보가 영어로 출력 되는 모습을 볼 수 있다.

1.3(2) When an ATM is configured bilingual, a user can choose if the information is to be displayed either English or Korean.

```
==========================================================================
2
Selected ATM [Bank: NH, SN: 222222]
==========================================================================
Choose Language
1: English
2: Korean
2
==========================================================================
언어가 한국어로 설정되었습니다.
==========================================================================
카드를 넣어 주세요.
22002200
==========================================================================
카드 비밀번호를 입력하세요.
==========================================================================
2200
==========================================================================
카드가 정상적으로 삽입되었습니다!
==========================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
==========================================================================
```

[REQ1.3.2] Bilingual ATM 2- English 선택 시 화면

```
2
Selected ATM [Bank: NHSerial: 222222]
==========================================================================
Choose Language
1: English
2: Korean
2
==========================================================================
언어가 한국어로 설정되었습니다.
==========================================================================
카드를 넣어 주세요.
220022
==========================================================================
카드 비밀번호를 입력하세요.
==========================================================================
2200
==========================================================================
카드가 정상적으로 삽입되었습니다!
==========================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
==========================================================================
```

[REQ1.3.3] Bilingual ATM- Korean 선택 시 화면

- Bilingual ATM을 선택하면, Choose Language 옵션에서 Korean/English를 각각 선택했을 때 해당되는 언어로 정보가 표시되는 것을 확인할 수 있다.

▼ (REQ1.4) A Bank deposits a certain amount of cashes to an ATM to serve users.

```
ATM::ATM(string bank_name, int serial, int type, int lang, int num_1000, int num_5000, int num_10000, int num_50000)
    primary_bank = FindBank(bank_name);
    if (!primary_bank) {
        cout << "Invalid bank name!" << endl;
        return;
    }
    serial_number = serial;
    ATM_type = type;
    ATM_lan_type = lang;
    ATM_1000_num = num_1000;
    ATM_5000_num = num_5000;
    ATM_10000_num = num_10000;
    ATM_50000_num = num_50000;
    SetATMFund(num_1000, num_5000, num_10000, num_50000);
    atm_list.push_back(this);
}
```

[REQ1.4.1] ATM constructor

- ATM을 생성할 때 50000원, 10000원, 5000원, 1000원 지폐를 설정해 넣어 둔다.

▼ (REQ1.5) A Bank can open an Account for a user with the necessary information to perform bank services.

1.5(1) (e.g.) Bank name (e.g, Kakao, Shinhan), User name, Account number (12-digit), Available funds, Transaction histories.

```cpp
Account* Bank::ValidateAccount(long long int account_number) {
    return Account::FindAccount(account_number);
}

Account* Bank::ValidateCard(Card* card) {
    Account* valid_account = card->GetAccount();
    if (FindBank(valid_account->GetBankName()) == nullptr) {
        return nullptr;
    }
    if (Account::FindAccount(valid_account->GetAccountNumber()) == nullptr) {
        return nullptr;
    }
    int inpassword;
    int try_num = 0;
    do {
        if (try_num == 0) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================" << endl;
                cout << "Enter the card password." << endl;
                cout << "================================================================" << endl;
            }
            else {
                cout << "================================================================" << endl;
                cout << "카드 비밀번호를 입력하세요." << endl;
                cout << "================================================================" << endl;
            }
        }
```

[REQ1.5.1] ValidateAcount 함수와 ValidateCard 함수

```cpp
        else {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================" << endl;
                cout << "Wrong card password. Please enter the card password again." << endl;
                cout << "================================================================" << endl;
            }
            else {
                cout << "================================================================" << endl;
                cout << "틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요." << endl;
                cout << "================================================================" << endl;
            }

        }
        cin >> inpassword;
        if (inpassword == card->GetPassword()) {
            return valid_account;
        }
        try_num++;
    } while (try_num < 3);
    return nullptr;
}
```

[REQ1.5.2] 사진 REQ1.5.1과 이어지는 ValidateCard 함수

- ValidateAcount는 계좌번호를 받고 해당 계좌번호를 가진 계좌를 반환한다. ValidateCard는 카드에 연결된 계좌를 반환한다. 반환된 계좌들은 거래 클래스들(Deposit, Withdrawal, Transfer)에서 접근이 가능하다.

▼(REQ1.6) A user may have multiple Accounts in a Bank.

```
================================ User ================================
Please select the user to use.
Available Users:
1: 은빈
2: 예진
3: 유진
4: 서연

0: Return to Main Menu
======================================================================
3
Selected User: 유진
User's Accounts:
: Bank Hana, Account No: 330033003300, Balance: 300000
: Bank Hana, Account No: 660066006600, Balance: 1000
```

[REQ1.6.1] User’s Account 조회

- User 유진이 같은 은행에서 두 개의 계좌를 가지고 있는 것을 확인할 수 있다.

▼(REQ1.7) A user may have Accounts in multiple Banks.

```
================================ User ================================
Please select the user to use.
Available Users:
1: 은빈
2: 예진
3: 유진
4: 서연

0: Return to Main Menu
======================================================================
2
Selected User: 예진
User's Accounts:
: Bank NH, Account No: 220022002200, Balance: 5000
: Bank Daegu, Account No: 550055005500, Balance: 1000000
```

[REQ1.7.1] User’s Account 조회

- User 예진이 서로 다른 은행에 각각 하나의 계좌를 가지고 있는 것을 확인할 수 있다.

▼ (REQ1.8) Each ATM have several types of transaction fees, and paid as follows

1.8(1) Deposit fee for non-primary banks: KRW 2,000; the fee is paid by inserting additional cash.

1.8(2) Deposit fee for primary banks: KRW 1,000; the fee is paid by inserting additional cash.

```
void Deposit::SetFee() {
    if (target_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else { transaction_fee = 2000; }
}
```

[REQ1.8.1] Deposit의 SetFee 함수

1.8(3) Withdrawal fee for a primary bank: KRW 1,000; the fee is paid from the withdrawal account.

1.8(4) Withdrawal fee for non-primary banks: KRW 2,000; the fee is paid from the withdrawal account.

```
void Withdrawal::SetFee() {
    if (source_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else {
        transaction_fee = 2000;
    }
}
```

[REQ1.8.2] Withdrawal의 SetFee 함수

1.8(5) Account transfer fee between primary banks: KRW 2,000; the fee is paid from the source account.

1.8(6) Account transfer fee between primary and non-primary banks: KRW 3,000; the fee is paid from the source account.

1.8(7) Account transfer fee between non-primary banks: KRW 4,000; the fee is paid from the source account.

1.8(8) Cash transfer fee to any bank type: KRW 1,000; the fee is paid by inserting additional cash.

```
void Transfer::SetFee() {
    if (transfer_type == 2) {
        if (source_account->GetBankName() == atm->GetPrimaryBankName() && target_account->GetBankName() == atm->GetPrimaryBankName()) {
            transaction_fee = 2000;
        }
        else if (source_account->GetBankName() != atm->GetPrimaryBankName() && target_account->GetBankName() != atm->GetPrimaryBankName()) {
            transaction_fee = 4000;
        }
        else {
            transaction_fee = 3000;
        }
    }
    else if (transfer_type == 1) {
        transaction_fee = 1000;
    }
}
```

[REQ1.8.3] Transfer의 SetFee 함수

- Deposit, Withdrawal, Transfer 클래스에 모두 SetFee 함수를 사용해 fee를 설정하였다.

▼ (REQ1.9) An admin can access the menu of “Transaction History” via an admin card (See REQ Display of Transaction History).

```
======================================================================
Please insert your card.
99999999
======================================================================
Admin Menu: Select an option
1. Transaction History

0. Return to Main Menu
```

[REQ1.9.1] Admin card 삽입한 후

- ATM의 card로 하드 코딩된 admin card(99999999)를 넣으면 관리자 모드로 접근하여 Transaction History 메뉴를 볼 수 있도록 하였다.

▼ (REQ1.10) An ATM only accepts and returns the following types of cashes and checks.

1.10(1) (Cash type) KRW 1,000, KRW 5,000, KRW 10,000, KRW 50,000
- When implementing the ATM, you need to take each denomination of bills into account. In other words, instead of representing the ATM’s remaining cash as a single number, it should be implemented in a way that allows you to know how many bills of each denomination are left.
- Therefore, all actions of inserting cash to the ATM are performed by specifying the number of bills for each denomination.

```
void ATM::UpdateATMFund(int a_one_thousand, int a_five_thousand, int a_ten_thousand, int a_fifty_thousand, int mark) {
    if (mark == 1) {
        ATM_1000_num += a_one_thousand;
        ATM_5000_num += a_five_thousand;
        ATM_10000_num += a_ten_thousand;
        ATM_50000_num += a_fifty_thousand;
    }
    else {
        ATM_1000_num -= a_one_thousand;
        ATM_5000_num -= a_five_thousand;
        ATM_10000_num -= a_ten_thousand;
        ATM_50000_num -= a_fifty_thousand;
    }
    SetATMFund(ATM_1000_num, ATM_5000_num, ATM_10000_num, ATM_50000_num);
    return;
}
```

[REQ1.10.1] ATM의 UpdateATMFund 함수

- (Cash type) ATM은 50000원, 10000원, 5000원, 1000원 지폐만 받아 ATM 자금을 업데이트한다.

1.10(2) (Check type) Any amount over KRW 100,000 check (e.g., KRW 100,000, 100,001, 234,567 are all valid checks)

```
void Deposit::GetCheck() {
    for (num_of_check = 0; num_of_check < 31; num_of_check++) {
        cin >> incheck;
        if (incheck == 0) {
            break;
        }
        else if (num_of_check >= 30) {
            atm->ShowError(OverDeposit);
            break;
        }
        else if (incheck < 100000) {
            atm->ShowError(UnusableCheck);
        }
        sum_of_check = sum_of_check + incheck;
    }
```

[REQ1.10.2] Deposit의 GetCheck 함수

- (Check type) 100000원 이상의 수표만 받을 수 있도록 설정했다.

▼ (REQ1.11) All accounts and ATMs shall be created and initialized during the program execution.

1.11(1) During the program execution, the necessary information to create accounts and ATMs shall be given from a user via console input (i.e., hard coding of account and ATM information is not allowed). The accounts and ATMs shall be created and initialized based on the user input.

```
================================ Bank ================================
Please select the bank to use.
Available Banks:
1: NH
2: Hana
3: 우리
4: Daegu

0: Return to Main Menu
======================================================================
1
Selected Bank: NH
============================== Account ===============================
Would you like to create an account with the selected bank?
1: Yes
0: Return to Bank Menu
======================================================================
1
============================== Account ===============================
Bank Name: NH
Enter User Name (0 to cancel): 은빈
Enter Account Number (12 digits, 0 to cancel): 110011001100
Enter Available Funds (0 to cancel): 100000
Enter Password (4 digits, 0 to cancel): 1100
======================================================================
Card Number: 11001100
======================================================================
Card created successfully for Account: 110011001100
Account created successfully with Bank: NH
```

[REQ1.11.1] Bank menu

- 사진에 보이는 것처럼 user name, account number 등을 직접 입력하여 계좌를 만들 수 있다. 카드 비밀번호는 계좌 비밀번호와 동일하게 설정된다.

```
================================ ATM =================================
Please select an option for ATM.
1: Create ATM
2: Use ATM

0: Return to Main Menu
======================================================================
1
Enter Primary Bank Name (0 to cancel): NH
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 1
Enter number of KRW 50000: 50
Enter number of KRW 10000: 2
Enter number of KRW 5000: 1
Enter number of KRW 1000: 0
ATM [Bank: NH, Serial: 111111] created successfully!
================================ ATM =================================
```

[REQ1.11.2] ATM menu

- 사진에 보이는 것처럼 primary bank name, type 등을 직접 입력하여 ATM을 만들 수 있다.

1.11(2) In other words, at the program's start, there must be a function that creates an ATM instance and a BANK instance. The initial state of the program may also include functions for selecting an ATM.

```
AdminCard created with Card Number: 99999999
============================== Main ==============================
Please select a task option.
1: Bank (Create or Use Bank)
2: User (Create or Use User)
3: ATM (Create or Use ATM)
4: Display all information

0: Exit Program
==================================================================
4
============================ Information ===========================
Displaying all Banks, Users, Accounts, and ATMs:
Banks:
Users:
Accounts:
ATMs:
Cards:
Card Number: 99999999, Admin: Yes, Linked Account: No Linked Account
==================================================================
```

[REQ1.11.3] Display All Information menu

- 프로그램을 시작하면 다음과 같이 콘솔창이 나온다. 이때 admin card 외에 아무 정보가 없는 것을 볼 수 있다.

```
============================ Information ===========================
Displaying all Banks, Users, Accounts, and ATMs:
Banks:
Bank Name: NH
Bank Name: Hana
Bank Name: 우리
Bank Name: Daegu
Users:
User Name: 은빈
User Name: 예진
User Name: 유진
User Name: 서연
Accounts:
Account No: 110011001100, User: 은빈, Balance: 100000
Account No: 220022002200, User: 예진, Balance: 5000
Account No: 330033003300, User: 유진, Balance: 300000
Account No: 440044004400, User: 서연, Balance: 30000
Account No: 550055005500, User: 예진, Balance: 1000000
Account No: 660066006600, User: 유진, Balance: 1000
ATMs:
ATM Serial Number: 111111
ATM Serial Number: 222222
ATM Serial Number: 333333
ATM Serial Number: 444444
ATM Serial Number: 555555
Cards:
Card Number: 99999999, Admin: Yes, Linked Account: No Linked Account
Card Number: 11001100, Admin: No, Linked Account: 110011001100
Card Number: 22002200, Admin: No, Linked Account: 220022002200
Card Number: 33003300, Admin: No, Linked Account: 330033003300
Card Number: 44004400, Admin: No, Linked Account: 440044004400
Card Number: 55005500, Admin: No, Linked Account: 550055005500
Card Number: 66006600, Admin: No, Linked Account: 660066006600
==================================================================
```

[REQ1.11.4] Display All Information menu

- 사용자가 원하는 대로 bank, user, account와 card, ATM을 추가한 모습이다.

2. ATM Session

▼ (REQ2.1) A session starts when a user inserts a card.

```
SetCardInserted(true);
if (language_option == 1) {
    cout << "================================================================" << endl;
    cout << "Card successfully inserted!" << endl;
}
else if (language_option == 2) {
    cout << "================================================================" << endl;
    cout << "카드가 정상적으로 삽입되었습니다!" << endl;
}
SessionStarts();
StartInterface(card);
```

[REQ2.1.1] ATM의 CardInsert 함수 부

- ATM의 CardInsert() 함수를 통해 카드를 받고, 삽입된 후에 세션을 시작하게 만들었다.

```
Please insert your card.
11001100
================================================================
Enter the card password.
================================================================
1100
================================================================
Card successfully inserted!
================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
================================================================
```

[REQ2.1.2] Validation이 된 경우 session 시작

```
Selected ATM [Bank: NH, SN: 111111]
This ATM only supprts English.
================================================================
Please insert your card.
43748383
================================error=================================
Error: Invalid card.
Your card is returned.
Returning to main ATM menu.
```

[REQ2.1.3] Validation이 되지 않은 경우 card 반환

▼ (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).

```
================================================================
카드가 정상적으로 삽입되었습니다!
================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
================================================================
1
================================================================
입금 유형을 선택하세요.
1. 현금
2. 수표

0. 취소
================================================================
0
```

[REQ2.2.1] 사용자 인터페이스 일부

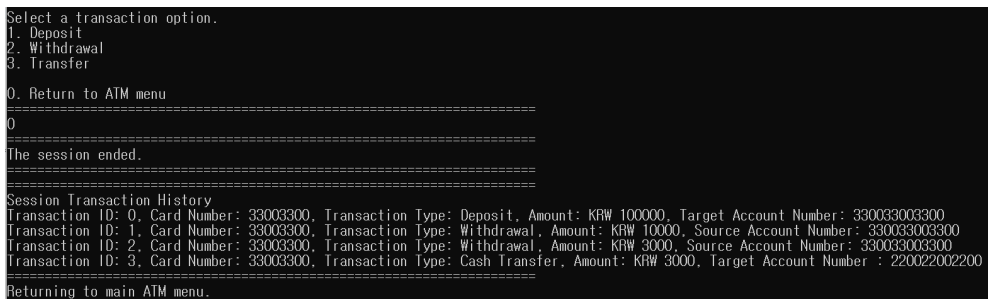
- 위 사진처럼 세션 도중 나타나는 모든 인터페이스에서, 0을 입력하면 세션을 끝낼 수 있도록 하였다.

[REQ2.2.2] exceptional conditions e.g. 비밀번호를 3번 틀린 경우

- 위 사진처럼 에러가 생기면 세션을 끝내도록 하였다. 더 자세한 에러 상황은 ‘9. Exception Handling’에서 확인할 수 있다.

▼ (REQ2.3) When a session ends, the summary of all transactions performed in a session must be displayed.

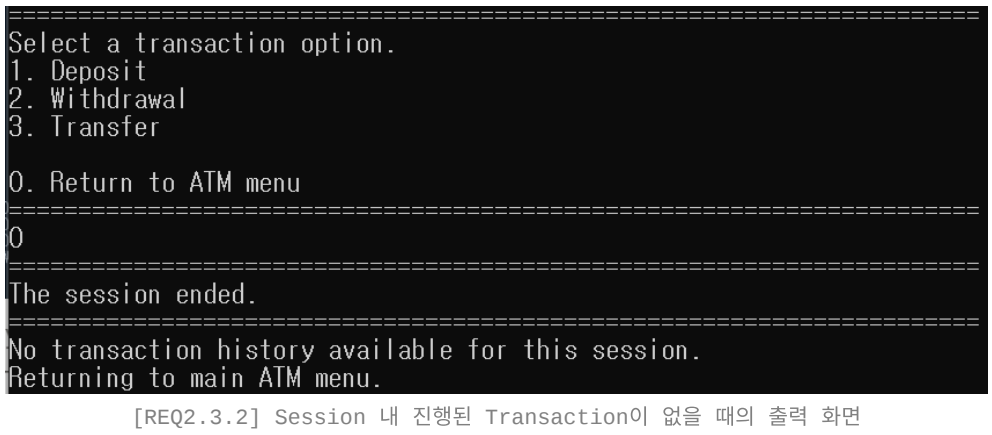
2.3(1)(e.g.) Account/card info, transaction types (deposit, transfer, withdrawal), and their amount, …



[REQ2.3.1] 4번의 Transaction 이후 Session 종료 시 출력 화면

- 새로운 session이 시작된 후의 transaction 내역을 ATM_history에 저장하고, 해당 session이 종료되었을 때 ATM_history에서 해당 session 내의 모든 transaction history를 출력하게 했다.

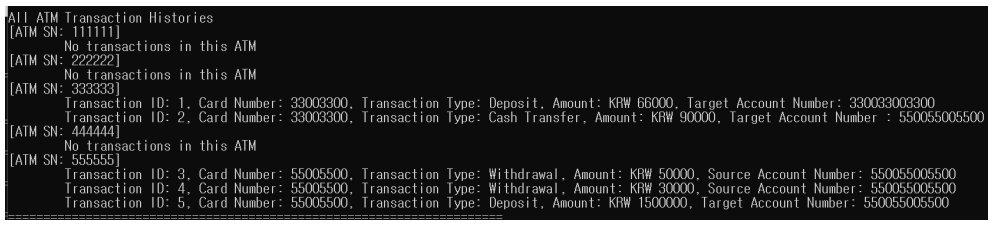
2.3(2) If no transactions are successfully completed during the session, it is acceptable not to print a summary.



[REQ2.3.2] Session 내 진행된 Transaction이 없을 때의 출력 화면

- session을 종료하였을 때 해당 session 내에서 성공적으로 완료된 거래 내역이 없으면 사진 [REQ2.3.2]와 같이 출력 되게 했다.
  - 언어가 영어로 설정되었을 때 출력 내용: No transaction history available for this session.
  - 언어가 한글로 설정되었을 때 출력 내용: 해당 세션에서의 거래 내역이 없습니다.

▼ (REQ2.4) Each transaction has a unique identifier across all sessions.



[REQ2.4.1] 모든 ATM의 거래 내역

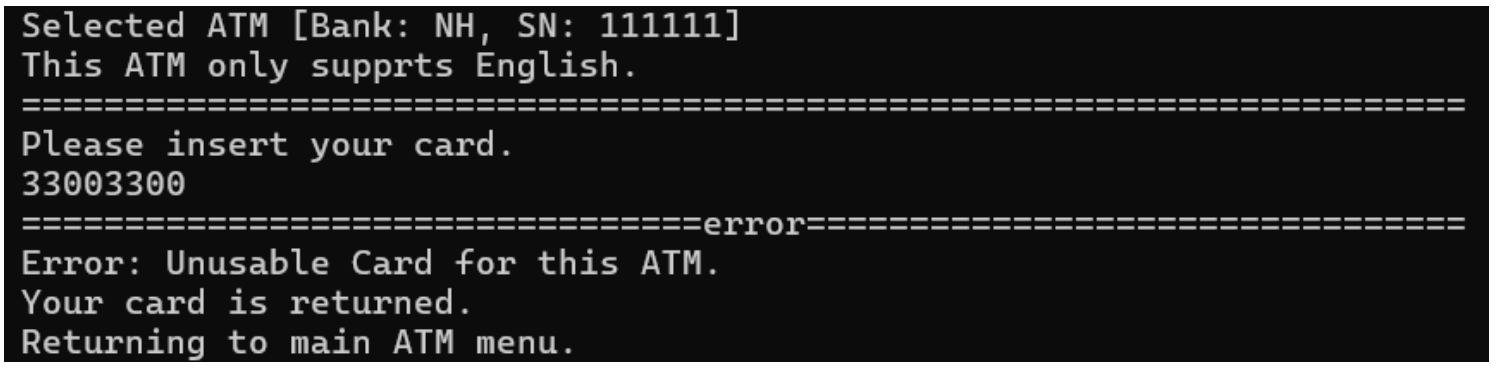
- ATM 종류와 상관없이 거래 순서대로 거래마다 고유한 번호가 설정된다.

3. User Authorization

▼ (REQ3.1) An ATM checks if the inserted card is valid for the current type of ATM.

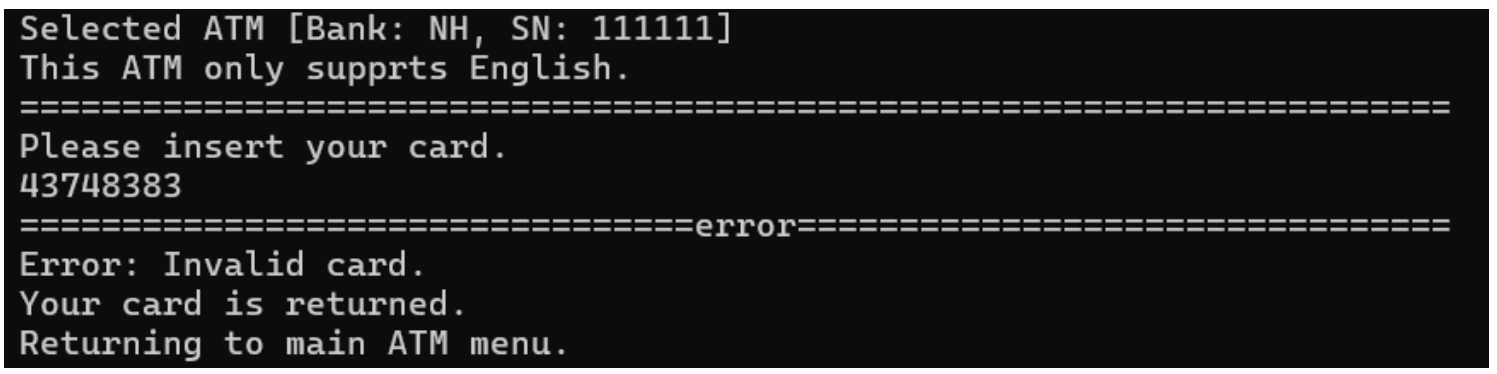
[REQ3.1.1] card valid check

- 현재의 ATM에 inserted card가 valid한지 확인하고 session을 시작하는 모습을 보인다.
- 현재 ATM의 type은 1으로 primary bank만 취급하지만, ATM과 card 모두 NH은행의 소유이므로 문제 없이 실행된다.



[REQ3.1.2] Single Bank ATM에 다른 은행 카드를 넣을 경우 invalid

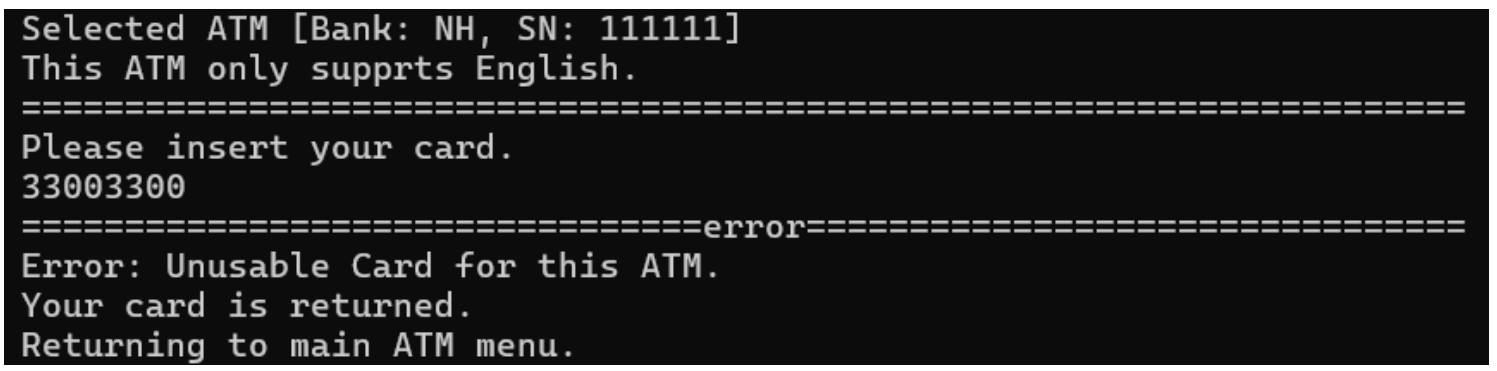
- type이 1인 ATM에 다른 은행 카드를 넣을 경우, unusable card 오류를 띄우며 작동하지 않는 모습을 볼 수 있다. ATM은 NH은행, card는 Hana 은행이다.



[REQ3.1.3] 존재하지 않는 카드를 넣을 경우 invalid

- 존재하지 않는 카드를 입력했을 경우에는 Invalid card 오류를 띄우며 작동하지 않는 모습을 보인다.

▼ (REQ3.2) If an invalid card is inserted, the ATM shall display an appropriate error message (e.g., Invalid Card).



[REQ3.2.1] card validation 과정에서 Single Bank ATM에 타 은행의 카드를 입력한 모습

- type이 1인 ATM에 다른 은행 카드를 넣어 invalid card를 감지한 경우, unusable card 오류를 띄우며 작동하지 않는 모습을 볼 수 있다. ATM은 NH은행, card는 Hana 은행이다.
- 존재하지 않는 카드를 입력하여 invalid card를 감지한 경우에는 Invalid card 오류를 띄우며 작동하지 않는 모습을 보인다.

▼ (REQ3.3) An ATM shall ask a user to enter the password (e.g., Enter Password), and verify if the password is correct.

3.3(1) An ATM does not maintain any user information, so the card and password information need to be passed to the bank; then, the bank verifies the password, and return the authorization result.

```
================================================================================
1
Selected ATM [Bank: NH, SN: 111111]
This ATM only suppports English.
================================================================================
Please insert your card.
11001100
================================================================================
Enter the card password.
================================================================================
1100
================================================================================
Card successfully inserted!
================================================================================
```

[REQ3.3.1] password 입력을 요청하는 모습

- console에서는 다음과 같이 카드의 validation 시 비밀번호 검증이 실행된다. ATM 화면에서는 위와 같이 보이지만, 실제로는 Bank class에서 카드 valid 여부와 비밀번호가 저장되어 있고 검증도 진행된다.

```
Account* Bank::ValidateCard(Card* card) {
    Account* valid_account = card->GetAccount();
    if (FindBank(valid_account->GetBankName()) == nullptr) {
        return nullptr;
    }
    if (Account::FindAccount(valid_account->GetAccountNumber()) == nullptr) {
        return nullptr;
    }
    int inpassword;
    int try_num = 0;
    do {
        if (try_num == 0) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================" << endl;
                cout << "Enter the card password." << endl;
                cout << "================================================================" << endl;
            }
            else {
                cout << "================================================================" << endl;
                cout << "카드 비밀번호를 입력하세요." << endl;
                cout << "================================================================" << endl;
            }
        }
        else {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================" << endl;
                cout << "Wrong card password. Please enter the card password again." << endl;
                cout << "================================================================" << endl;
            }
            else {
                cout << "================================================================" << endl;
                cout << "틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요." << endl;
                cout << "================================================================" << endl;
            }
        }
        cin >> inpassword;
        if (inpassword == card->GetPassword()) {
            return valid_account;
        }
        try_num++;
    } while (try_num < 3);
    return nullptr;
}
```

[REQ3.3.2] password 검증이 진행되는 코드

- session start 전 card가 insert되었을 때와 실행되는 Bank class의 ValidateCard 함수는 계좌 정보와 비밀번호 정보를 가지고 있지 않은 ATM class를 대신하여 카드의 존재 여부와 카드 비밀번호를 검증한다. 검증에 성공하였을 경우 계좌 정보를, 실패했을 경우 nullptr을 return하여 허가 결과를 return한다.

▼ (REQ3.4) If the entered password is incorrect, the ATM shall display an appropriate error message (e.g., Wrong Password).

```
================================================================================
Please insert your card.
22002200
================================================================================
Enter the card password.
================================================================================
1234
================================================================================
Wrong card password. Please enter the card password again.
================================================================================
1010
================================================================================
Wrong card password. Please enter the card password again.
================================================================================
1111
==================================error=========================================
Error: Incorrect password.
================================================================================
```

[REQ3.4.1] password를 3회 연속 틀린 모습

- card 검증 과정에서 잘못된 비밀번호를 입력하였을 경우, Wrong card password. Please enter the card password again.이라는 오류를 띄운다. 세 번째로 비밀번호를 잘못 입력한 경우에는 Error: Incorrect password라는 오류를 띄운다.

```
11111111
================================================================================
카드 비밀번호를 입력하세요.
================================================================================
1110
================================================================================
틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요.
================================================================================
1110
================================================================================
틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요.
================================================================================
1111
================================================================================
카드가 정상적으로 삽입되었습니다!
================================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
================================================================================
```

[REQ3.4.2] password를 틀렸으나 3회 이내에 올바를 password를 입력한 모습

- 비밀번호를 잘못 입력했다고 하더라도 연속으로 3회 틀리지 않을 경우, 카드가 성공적으로 삽입되었다는 메시지가 출력되고 거래 유형을 선택할 수 있다.

▼ (REQ3.5) If a user enters wrong passwords 3 times in a row, a session is aborted, and return the card to the user.

```
================================================================================
Please insert your card.
11001100
================================================================================
Enter the card password.
================================================================================
4387
================================================================================
Wrong card password. Please enter the card password again.
================================================================================
3027
================================================================================
Wrong card password. Please enter the card password again.
================================================================================
6738
==================================error=========================================
Error: Incorrect password.
Your card is returned.
Returning to main ATM menu.
```

[REQ 3.5.1.] password를 3회 연속 틀린 모습

- card 검증 과정에서 잘못된 비밀번호를 연속해서 3번 입력한 경우, session이 시작되지 않고 카드를 user에게 돌려준다.

4. Deposit

▼ (REQ4.1) An ATM shall take either cash or check from a user.

```
================================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
================================================================================
1
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
```

[REQ4.1.1] ATM 3, Account 6

- transaction 중 deposit을 선택한 경우, cash deposit이나 check deposit 중에서 deposit 방식을 선택할 수 있다.

4.1(1) The number of bills is entered separately for each denomination.

```
================================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 0
Enter number of KRW 10000: 0
Enter number of KRW 5000: 2
Enter number of KRW 1000: 0
================================================================================
```

[REQ4.1.2] ATM 3, Account 6

- cash deposit을 선택한 경우, 각 지폐 종류별로 몇 장을 deposit할 것인지 입력받는다.

▼(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.

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
1
==========================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 1
Enter number of KRW 10000: 2
Enter number of KRW 5000: 3
Enter number of KRW 1000: 50
==========================================================================
==================================error===================================
Error: Deposit limit exceeded.
==========================================================================
The session ended.
==========================================================================
```

[REQ4.2.1] 한 session에서 지폐 50장을 초과한 cash deposit 시도한 경우

- (Cash) 한 session 내에서 50장을 초과한 cash deposit이 일어날 경우 오류를 보이며 거래가 시행되지 않고 session이 종료된다.

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
2
==========================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
==========================================================================
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
```

```
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000

100000
100000
100000
100000
100000
100000
100000
100000
100000
100000

100000
==========================================error==========================================
Error: Deposit limit exceeded.
=========================================================================================
The session ended.
=========================================================================================
```

[REQ4.2.2] 한 session에서 수표 30장을 초과한 check deposit을 시도한 경우

- (Check) 한 session 내에서 30장을 초과한 check deposit이 일어날 경우 오류를 보이며 거래가 시행되지 않고 session이 종료된다.

▼ (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).

- (Cash)

```
=====================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
=====================================================================
1
=====================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 0
Enter number of KRW 10000: 0
Enter number of KRW 5000: 2
Enter number of KRW 1000: 0
=====================================================================
=====================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
=====================================================================
1000
=====================================================================
Deposit proceeded successfully.
=====================================================================
```

[REQ4.3.1] ATM 3, Account 6

```
========================= ATM Snapshots ============================
ATM [SN: 333333] Remaining Cash: {Total: 66000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 3, KRW 1000: 51}
=====================================================================
========================= Account Snapshots ========================
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 11000
=====================================================================
```

[REQ4.3.2] ATM 3, Account 6

- (Cash) account 6은 초기에 설정된 잔액이 1000원이었으나 10000원의 check deposit이 끝나고 11000원으로 계좌 잔액이 업데이트된 것을 알 수 있다.

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
2
==========================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
==========================================================================
100000
0
==========================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
==========================================================================
1000
==========================================================================
Deposit proceeded successfully.
==========================================================================
```

[REQ4.3.3] ATM 3, Account 6

```
========================== ATM Snapshots ===========================
ATM [SN: 333333] Remaining Cash: {Total: 56000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 1, KRW 1000: 51}
======================================================================
========================= Account Snapshots ==========================
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 101000
======================================================================
```

[REQ4.3.4] ATM 3, Account 6

- (Check) account 6은 초기에 설정된 잔액이 1000원이었으나 100000원의 check deposit이 끝나고 101000원으로 계좌 잔액이 업데이트 된 것을 알 수 있다.

▼ (REQ4.4) Some deposit fee may be charged (See REQ in System Setup).

```
void Deposit::SetFee() {
    if (target_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else { transaction_fee = 2000; }
}
```

[RE4.4.1] deposit에서 사용되는 SetFee함수

- deposit에서는 card에 연결된 account의 은행이 ATM의 주거래 은행과 일치할 경우 1000원을, 일치하지 않을 경우 2000원의 수수료를 받는다.

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
2
==========================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
==========================================================================
100000
0
==========================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
==========================================================================
1000
==========================================================================
Deposit proceeded successfully.
==========================================================================
```

[REQ4.4.2] ATM 3, Account 6

- ATM 3과 Account 6 모두 Hana 은행의 것으로 account가 ATM의 주거래 은행이기 때문에 1000원의 수수료가 부과된다.

```
================================================================================
입금 유형을 선택하세요.
1. 현금
2. 수표

0. 취소
================================================================================
2
================================================================================
수표는 최대 30장까지 넣을 수 있습니다.
수표 입력을 중지하려면 '0'을 눌러 주세요.
================================================================================
100000
0
================================================================================
수수료를 현금으로 납부해 주세요.
(주거래 은행 예치 요금: 1000원, 주거래 은행 외 예치 요금: 2000원)
================================================================================
2000
================================================================================
입금이 정상적으로 진행되었습니다.
================================================================================
```

[REQ4.4.3] ATM 2, Account 5

- ATM 2는 NH 은행이고 Account 5는 Daegu 은행이므로, account가 ATM의 주거래 은행이 아니기 때문에 2000원의 수수료가 부과된다.

4.4.(1) In case of a check deposit, a fee must be served as cash.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
2
================================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
================================================================================
100000
0
================================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
================================================================================
1000
================================================================================
Deposit proceeded successfully.
================================================================================
```

[REQ4.4.4] ATM 3, Account 6

```
========================== ATM Snapshots ===========================
ATM [SN: 333333] Remaining Cash: {Total: 56000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 1, KRW 1000: 51}
====================================================================
========================= Account Snapshots ========================
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 101000
====================================================================
```

[REQ4.4.5] ATM 3, Account 6

- check deposit의 경우에도, 수수료는 지폐로 받는 것을 확인할 수 있다. 위 화면에서는 1000원의 수수료를 지폐 금액으로 받는다.

4.4(2) The deposit amount and the fee must be entered separately, with one entry for the deposit and another for the fee.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
1
================================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 0
Enter number of KRW 10000: 0
Enter number of KRW 5000: 2
Enter number of KRW 1000: 0
================================================================================
================================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
================================================================================
1000
================================================================================
Deposit proceeded successfully.
================================================================================
```

[REQ4.4.6] ATM 3, Account

- cash deposit의 경우, deposit하고자 하는 지폐 종류별로 개수를 먼저 입력하고 나서 수수료를 따로 입력받는다.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
2
================================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
================================================================================
100000
0
================================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
================================================================================
1000
================================================================================
Deposit proceeded successfully.
================================================================================
```

[REQ4.4.7] ATM 3, Account 6

- check deposit의 경우, deposit하고자 하는 check 금액을 먼저 입력하고 나서 수수료를 따로 입력받는다.

▼ (REQ4.5) The deposited cash increase available cash in ATM that can be used by other users.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
1
================================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 0
Enter number of KRW 10000: 0
Enter number of KRW 5000: 2
Enter number of KRW 1000: 0
================================================================================
================================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
================================================================================
1000
================================================================================
Deposit proceeded successfully.
================================================================================
```

[REQ4.5.1] ATM 3, Account 6

```
========================= ATM Snapshots ==========================
ATM [SN: 333333] Remaining Cash: {Total: 66000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 3, KRW 1000: 51}
================================================================================
======================== Account Snapshots =======================
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 11000
================================================================================
```

[REQ4.5.2] ATM 3, Account 6

- 초기에 설정된 ATM 333333은 지폐 종류별 개수가 50000원, 10000원, 5000원, 1000원 순으로 0, 0, 1, 50개였다.
- 5000원 권을 2개 넣어 10000원의 cash deposit이 끝난 이후에, primary bank에서 deposit할 시 받는 수수료 1000원을 포함하여 ATM 지폐 개수는 50000원, 10000원, 5000원, 1000원 순으로 0, 0, 3, 51개로 업데이트가 일어난 것을 볼 수 있다.

▼ (REQ4.6) The deposited check does not increase available cash in ATM that can be used by other users.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
2
================================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
================================================================================
100000
0
================================================================================
Please put in the appropriate fee as bills.
(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)
================================================================================
1000
================================================================================
Deposit proceeded successfully.
================================================================================
```

[REQ4.6.1] ATM 3, Account 6

```
============================ ATM Snapshots ============================
ATM [SN: 333333] Remaining Cash: {Total: 56000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 1, KRW 1000: 51}
=======================================================================
========================= Account Snapshots ===========================
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 101000
=======================================================================
```

[REQ4.6.2] ATM 3, Account 6

- 초기에 설정된 ATM 333333은 지폐 종류별 개수가 50000원, 10000원, 5000원, 1000원 순으로 0,0,1,50개였다.
- 100000원의 check deposit이 끝난 이후에 primary bank에서 deposit할 시 받는 수수료 1000원만큼만 ATM 지폐 개수에 업데이트가 일어나는 것을 볼 수 있다.

5. Withdrawal

▼ (REQ5.1) An ATM shall ask a user to enter the amount of fund to withdraw.

5.1(1) The user does not manually input the number of each denomination. Instead, the user only enters the desired withdrawal amount, and the ATM will dispense the cash using the fewest number of possible bills (which means using the highest denomination bills as much as possible)

```
=======================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
=======================================================================
17000
=======================================================================
You have successfully withdrawn: KRW 17000
Withdrawal fee: KRW 1000
Fees have been paid from the withdrawal account.
=======================================================================
```

[REQ5.1.1] ATM 4, Account 4

- 사용자가 출금을 원하는 금액을 직접 입력할 수 있게 하였다.

5.1(2) For example, when user withdrawals KRW 17,000, the ATM dispenses:

1 bill of KRW 10,000

1 bill of KRW 5,000

2 bills of KRW 1,000

```
============================ ATM Snapshots ============================
ATM [SN: 444444] Remaining Cash: {Total: KRW 330000, KRW 50000: 5, KRW 10000: 5, KRW 5000: 5, KRW 1000: 5}
=======================================================================
========================= Account Snapshots ===========================
Account [Bank: 우리, No: 4400044004400, Owner: 서연] Balance: 30000
=======================================================================
```

[REQ5.1.2] ATM 4, Account 4

```
=======================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
=======================================================================
17000
=======================================================================
You have successfully withdrawn: KRW 17000
Withdrawal fee: KRW 1000
Fees have been paid from the withdrawal account.
=======================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
=======================================================================
/
============================ ATM Snapshots ============================
ATM [SN: 444444] Remaining Cash: {Total: 313000, KRW 50000: 5, KRW 10000: 4, KRW 5000: 4, KRW 1000: 3}
=======================================================================
========================= Account Snapshots ===========================
Account [Bank: 우리, No: 4400044004400, Owner: 서연] Balance: 12000
=======================================================================
```

[REQ5.1.3] ATM 4, Account 4

- 17000원을 입력했을 때 10000원권 1장, 5000원권 1장, 1000원권 2장이 ATM에서 출금된 것을 확인할 수 있다.

▼ (REQ5.2) An ATM shall display an appropriate error message if there is insufficient fund in the account or insufficient cash in the ATM.

```
=======================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of
 KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
=======================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
=======================================================================
50000
=================================error=================================
Error: Insufficient ATM balance.
=======================================================================
The session ended.
=======================================================================
```

[REQ5.2.1] ATM 2

- 총 37000원이 있는 ATM 2에서 50000원 출금을 시도했을 때 에러가 난다.

```
========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
========================================================================
200000
==================================error=================================
Error: Insufficient account balance.
========================================================================
The session ended.
========================================================================
```

[REQ5.2.2] Account 1

- 총 100000원이 있는 Account 1에서 200000원 출금을 시도했을 때 에러가 난다.

▼ (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).

```
========================= ATM Snapshots ============================
ATM [SN: 111111] Remaining Cash: {Total: 2525000, KRW 50000: 50, KRW 10000: 2, KRW 5000: 1, KRW 1000: 0}
========================================================================
========================= Account Snapshots ========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 100000
========================================================================
```

[REQ5.3.1] ATM 1, Account 1

```
========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
========================================================================
10000
========================================================================
You have successfully withdrawn: KRW 10000
Transaction fee: KRW 1000
Fees have been paid from the withdrawal account.
========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
========================================================================
/
============================ ATM Snapshots =============================
ATM [SN: 111111] Remaining Cash: {Total: 2515000, KRW 50000: 50, KRW 10000: 1, KRW 5000: 1, KRW 1000: 0}
========================================================================
========================= Account Snapshots ========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 89000
========================================================================
```

[REQ5.3.2] ATM 1, Account 1

- 출금 후 잔액이 ATM 1과 Account 1에 잘 반영된 것을 확인할 수 있다.

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

```
void Withdrawal::SetFee() {
    if (source_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else {
        transaction_fee = 2000;
    }
}
```

[REQ5.4.1] Withdrawal에서 사용되는 SetFee 함수

```
if (ATM::GetLanguageOption() == 1) {
    cout << "========================================================================" << endl;
    cout << "You have successfully withdrawn: KRW " << transaction_amount << endl;
    cout << "Withdrawal fee: KRW " << transaction_fee << endl;
    cout << "Fees have been paid from the withdrawal account." << endl;
}
else {
    cout << "========================================================================" << endl;
    cout << "성공적으로 출금되었습니다: " << transaction_amount << " 원" << endl;
    cout << "출금 수수료: " << transaction_fee << " 원" << endl;
    cout << "수수료가 출금 계좌에서 차감되었습니다." << endl;
}

source_account->UpdateAccountMoney(transaction_amount + transaction_fee, 0);
atm->UpdateATMFund(one_thousand, five_thousand, ten_thousand, fifty_thousand, 0);
withdrawal_count++;
return;
```

[REQ5.4.2] Withdrawal의 WithdrawalInprogress 함수

- withdrawal에서는 card에 연결된 account의 은행이 ATM의 주거래 은행과 일치할 경우 1000원을, 일치하지 않을 경우 2000원의 수수료를 받는다.

▼ (REQ5.5) The cash withdrawal lowers available cash in the ATM that can be used by other users.

```
========================= ATM Snapshots ============================
ATM [SN: 111111] Remaining Cash: {Total: 2525000, KRW 50000: 50, KRW 10000: 2, KRW 5000: 1, KRW 1000: 0}
========================================================================
========================= Account Snapshots ========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 100000
========================================================================
```

[REQ5.5.1] ATM 1, Account 1

```
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
10000
==========================================================================
You have successfully withdrawn: KRW 10000
Transaction fee: KRW 1000
Fees have been paid from the withdrawal account.
==========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
==========================================================================
/
=========================== ATM Snapshots ============================
ATM [SN: 111111] Remaining Cash: {Total: 2515000, KRW 50000: 50, KRW 10000: 1, KRW 5000: 1, KRW 1000: 0}
==========================================================================
======================== Account Snapshots ========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 89000
==========================================================================
```

[REQ5.5.2] ATM 1, Account 1

- ATM 내의 지폐가 출금되어 사용 가능한 지폐의 수가 줄어들었다.

▼ (REQ5.6) The maximum number of withdrawals per each session is 3.

5.6(1) 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.

```
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
10000
==========================================================================
You have successfully withdrawn: KRW 10000
Withdrawal fee: KRW 1000
Fees have been paid from the withdrawal account.
==========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
==========================================================================
2
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
1 withdrawals have been made in this session.
==========================================================================
```

[REQ5.6.1] ATM 1, Account 1

```
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
2 withdrawals have been made in this session.
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
5000
==========================================================================
You have successfully withdrawn: KRW 5000
Withdrawal fee: KRW 1000
Fees have been paid from the withdrawal account.
==========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
==========================================================================
2
==========================================================================
Session limit reached.
Please start a new session for further withdrawals.
==========================================================================
The session ended.
==========================================================================
```

[REQ5.6.2] ATM 1, Account 1

- withdrawal을 선택했을 때 해당 세션에서 몇 번의 출금이 진행됐는지 공지한다. 같은 세션에서 4번째 출금을 시도하면 오류 메시지와 함께 자동으로 세션이 끝난다.

▼ (REQ5.7) The maximum amount of cash withdrawal per transaction is KRW 500,000.

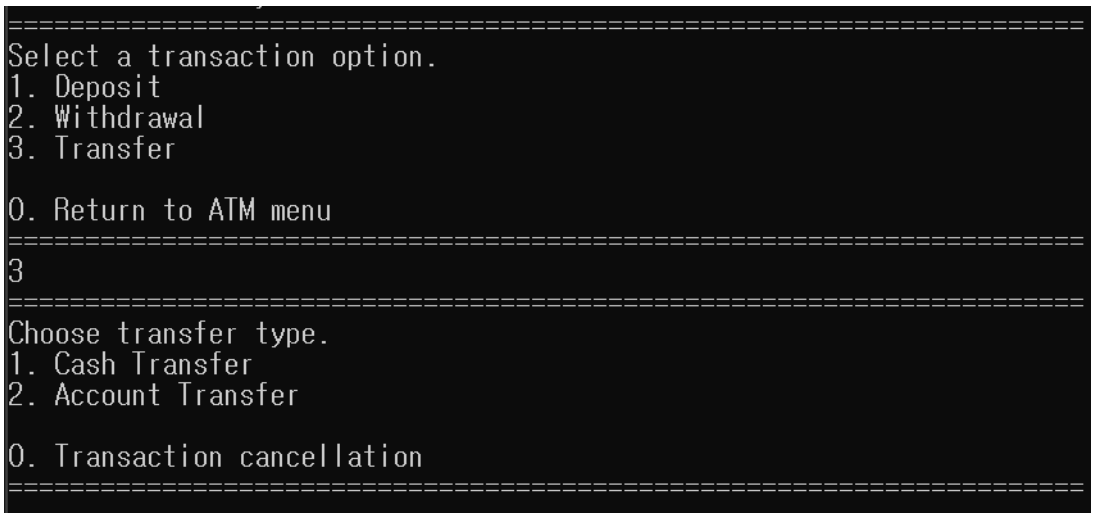
```
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
700000
================================error=================================
Error: Single deposit limit of 500,000 won exceeded.
==========================================================================
```

[REQ5.7.1] ATM 1, Account 1

- 최대 출금 가능 금액인 500000원을 초과하는 금액인 700000원을 출금하려고 하면 오류가 뜬다.

6. Transfer

▼ (REQ6.1) An ATM shall ask a user to choose the transfer types either cash transfer or account fund transfer.



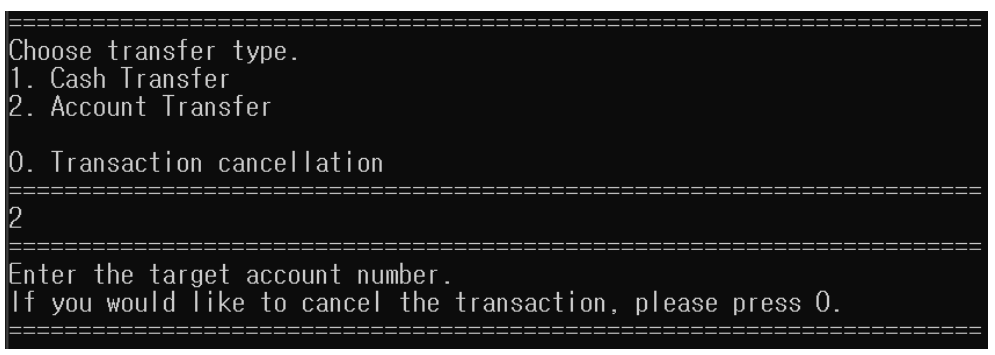
[REQ6.1.1] Transfer 유형 선택

- Transfer 거래를 선택하면, Transfer 유형(Cash transfer/Acconut transfer)을 선택하는 화면이 등장한다.

▼ (REQ6.2) For both cash and account transfers, an ATM shall ask the destination account number where the fund is to be transferred.

[REQ6.2.1] Cash transfer 진행 시 target account number 입력

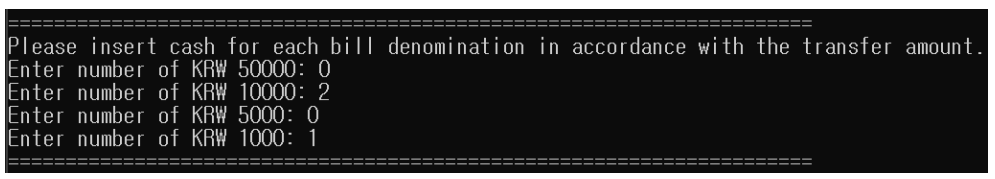
- Cash Transfer를 선택했을 때, destination account number를 입력받는다.



[REQ6.2.2] Account transfer 진행 시 target account number 입력

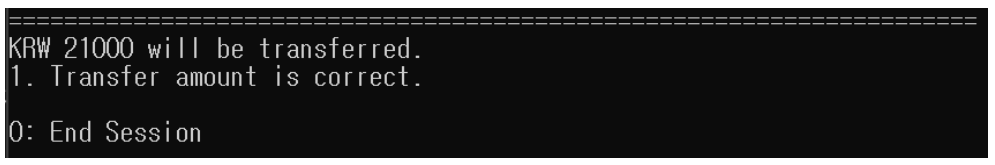
- Account Transfer를 선택한 경우에도, target account number를 입력받는다.

▼ (REQ6.3) For cash transfer, an ATM shall ask the user to insert the cash and transaction fees. After all the cash has been inserted, the ATM shall verify the amount to be transferred, excluding the transaction fee. All inserted cash, minus the transaction fee, shall be transferred.



[REQ6.3.1] ATM 2, target account 4

[REQ6.3.2] cash transfer 진행 시 fee를 받는 모습



[REQ6.3.3] cash transfer 진행 시 최종 이체 금액 확인

- ATM이 이체할 지폐 수와 fee를 따로 입력받는다. 현금 이체 전, ATM은 이체될 금액을 확인시켜 준다.

▼(REQ6.4) For account transfer, an ATM shall ask the source account number, and the amount of fund to be transferred. (If it is assumed that the source account has already been accessed at the start of the session using a card or other means, there is no need to ask again.)

```
==========================================================================
Choose transfer type.
1. Cash Transfer
2. Account Transfer

0. Transaction cancellation
==========================================================================
2
==========================================================================
Enter the target account number.
If you would like to cancel the transaction, please press 0.
==========================================================================
220022002200
==========================================================================
Enter amount to transfer.
If you would like to cancel the transaction, please press 0.
==========================================================================
10000
==========================================================================
```

[REQ6.4.1] Transfer 인터페이스 중 일부

- 이체를 받을 계좌번호를 입력하고 이체할 금액을 입력한다. (세션 시작 시 삽입한 카드와 연결된 계좌에서 이체된다.)

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

```
void Transfer::SetFee() {
    if (transfer_type == 2) {
        if (source_account->GetBankName() == atm->GetPrimaryBankName() && target_account->GetBankName() == atm->GetPrimaryBankName()) {
            transaction_fee = 2000;
        }
        else if (source_account->GetBankName() != atm->GetPrimaryBankName() && target_account->GetBankName() != atm->GetPrimaryBankName()) {
            transaction_fee = 4000;
        }
        else {
            transaction_fee = 3000;
        }
    }
    else if (transfer_type == 1) {
        transaction_fee = 1000;
    }
}
```

[REQ6.5.1] SetFee() code

- transfer에서는 cash transfer와 account transfer의 fee가 다르게 설정된다. account transfer에서는 주거래 은행 계좌 간 이체일 때 2000원, 주거래 은행 계좌와 주거래 은행 외 계좌 간 이체일 때 3000원, 주거래 은행 외 계좌 간 거래에서 4000원이 된다. cash transfer에서는 1000원을 받는다.

```
==========================================================================
Please put in the appropriate fee as bills.
(Cash transfer fee between primary banks: KRW 2000, Cash transfer fee between
primary and non-primary banks: KRW 3000,
Cash transfer fee between non-primary banks: KRW 4000)
==========================================================================
```

[REQ6.5.2] account transfer 진행 시 fee 안내

- account transfer의 경우에서도 위 사진처럼 fee를 입력받기 전 fee에 대해 안내한다.

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

```
========================= ATM Snapshots ==========================
ATM [SN: 111111] Remaining Cash: {Total: KRW 2525000, KRW 50000: 50, KRW 10000: 2, KRW 5000: 1, KRW 1000: 0}
ATM [SN: 222222] Remaining Cash: {Total: KRW 108000, KRW 50000: 1, KRW 10000: 5, KRW 5000: 1, KRW 1000: 3}
ATM [SN: 333333] Remaining Cash: {Total: KRW 55000, KRW 50000: 0, KRW 10000: 0, KRW 5000: 1, KRW 1000: 50}
ATM [SN: 444444] Remaining Cash: {Total: KRW 330000, KRW 50000: 5, KRW 10000: 5, KRW 5000: 5, KRW 1000: 5}
ATM [SN: 555555] Remaining Cash: {Total: KRW 155000, KRW 50000: 2, KRW 10000: 3, KRW 5000: 4, KRW 1000: 5}
==================================================================
```

[REQ6.6.1] ATM 2, target account 4에 cash transfer한 후

- ATM 2에 처음에 50000원권 0장, 10000원권 3장, 5000원권 1장, 1000원권 2장이 있었는데, target account 4에 cash transfer로 50000원권 1장, 10000원권 2장을 보낸 후에 ATM 2의 자금이 업데이트되었다.

▼(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).

```
======================== Account Snapshots ========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 100000
Account [Bank: NH, No: 220022002200, Owner: 예진] Balance: 5000
Account [Bank: Hana, No: 330033003300, Owner: 유진] Balance: 300000
Account [Bank: 우리, No: 440044004400, Owner: 서연] Balance: 100000
Account [Bank: Daegu, No: 550055005500, Owner: 예진] Balance: 1000000
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 1000
==================================================================
```

[REQ6.7.1] ATM 2, target account 4에 cash transfer한 후

- target account 4에 70000원이 이체되었다.

```
========================= Account Snapshots =========================
Account [Bank: NH, No: 110011001100, Owner: 은빈] Balance: 67000
Account [Bank: NH, No: 220022002200, Owner: 예진] Balance: 5000
Account [Bank: Hana, No: 330033003300, Owner: 유진] Balance: 300000
Account [Bank: 우리, No: 440044004400, Owner: 서연] Balance: 100000
Account [Bank: Daegu, No: 550055005500, Owner: 예진] Balance: 1000000
Account [Bank: Hana, No: 660066006600, Owner: 유진] Balance: 31000
=====================================================================
```

[REQ6.7.2] ATM 1, source account 1에서 target account 6으로 account transfer한 후

- source account 1에서 33000원(30000원+3000원)이 출금되고, target account 6에서 30000원이 이체를 받았다.

7. Display of Transaction History (Admin Menu)

▼ (REQ7.1) When a session is started by an admin by inserting an admin card, an ATM displays a menu of “Transaction History” only.

```
1
Selected ATM [Bank: NH, SN: 111111]
This ATM only suppports English.
================================================================================
Please insert your card.
99999999
================================================================================
Admin Menu: Select an option
1. Transaction History

0. Return to Main Menu
```

[REQ7.1.1] ATM 1, admin card inserted

- admin card를 넣은 경우 ATM의 콘솔에 Transaction History와 cancel만 선택할 수 있는 화면을 띄운다.

▼ (REQ7.2) When the “Transaction History” menu is selected, an ATM displays the information of all transactions from all users since the system started.

7.2(1) Transaction ID, Card Number, Transaction Types, Amount, other transaction-specific information

7.2(2) 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).

```
====================================================================
All ATM Transaction Histories
[ATM SN: 111111]
        Transaction ID: 2, Card Number: 11001100, Transaction Type: Account Transfer, Amount: KRW 30000, Source Account Number: 110011001100, Target Account Number: 660066006600
[ATM SN: 222222]
        Transaction ID: 1, Card Number: 55005500, Transaction Type: Cash Transfer, Amount: KRW 70000, Target Account Number : 440044004400
[ATM SN: 333333]
        No transactions in this ATM
[ATM SN: 444444]
        No transactions in this ATM
[ATM SN: 555555]
        Transaction ID: 3, Card Number: 33003300, Transaction Type: Deposit, Amount: KRW 82000, Target Account Number: 330033003300
        Transaction ID: 4, Card Number: 33003300, Transaction Type: Deposit, Amount: KRW 100000, Target Account Number: 330033003300
        Transaction ID: 5, Card Number: 33003300, Transaction Type: Account Transfer, Amount: KRW 10000, Source Account Number: 330033003300, Target Account Number: 440044004400
        Transaction ID: 6, Card Number: 44004400, Transaction Type: Withdrawal, Amount: KRW 10000, Source Account Number: 440044004400
        Transaction ID: 7, Card Number: 44004400, Transaction Type: Withdrawal, Amount: KRW 4000, Source Account Number: 440044004400
====================================================================
Transaction history saved to transaction_history.txt
Returning to main ATM menu.
```

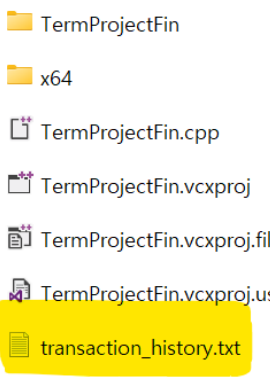
[REQ7.2.1] Admin card inserted, Transaction History menu selected

- Transaction History 메뉴를 선택한 경우, 여러 session에 걸친 모든 거래 기록이 출력된다.
- Transaction ID, Card Number, Transaction Types, Amount는 모든 거래 종류에서 출력된다.
- 거래별로 deposit과 cash transfer에서는 target account number이, withdrawal에서는 source account number이, account transfer에서는 target account number과 source account number이 추가로 출력된다.
- ATM에서 거래가 진행되지 않은 경우에는 No transactions in this ATM.이라는 문구를 출력한다.

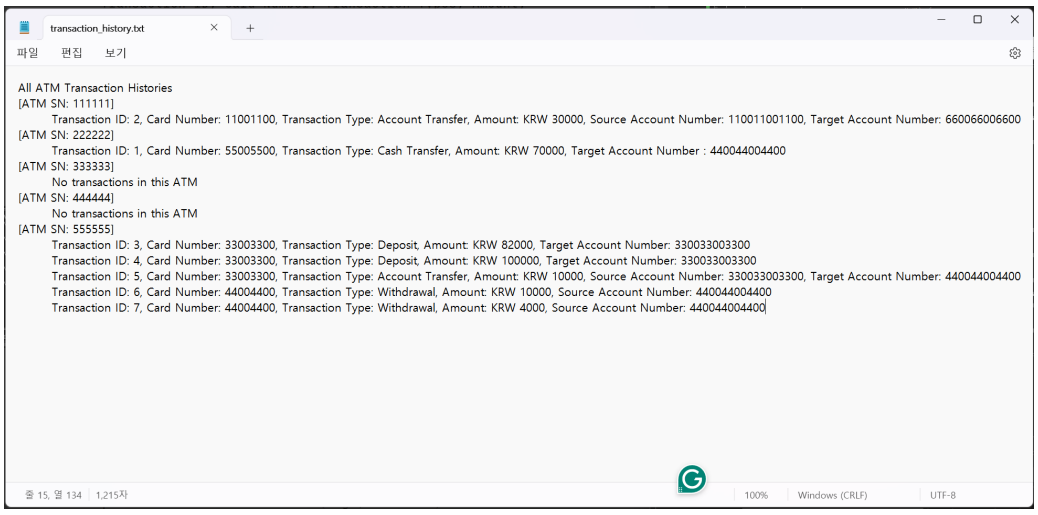
▼ (REQ7.3) The “Transaction History” information shall be outputted to the external file (e.g., txt file).

```cpp
void ATM::TransactionHistorytoFile() {
    ofstream file("transaction_history.txt");
    if (file.is_open()) {
        for (size_t i = 0; i < ATM_history.size(); i++) {
            for (size_t j = 0; j < ATM_history[i].size(); j++) {
                file << ATM_history[i][j] << endl;
            }
        }
        file.close();
        if (language_option == 1) { cout << "Transaction history saved to transaction_history.txt" << endl; }
        else if (language_option == 2) { cout << "거래 내역이 transaction_history.txt에 저장되었습니다." << endl; }
    }
    else {
        ShowError(FileError);
        return;
    }
}
```

[REQ7.3.1]Transaction History txt file을 저장하는 code



[REQ7.3.2]transaction_history.txt가 저장된 모습

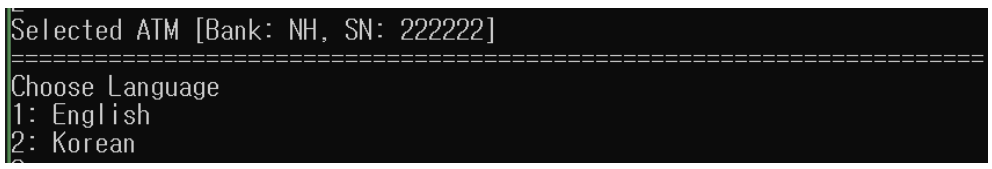


[REQ7.3.3]transaction_history.txt에 저장된 내용

- Transaction History가 실행되고 나면 외부의 txt 파일로 transaction History 내용이 저장된다.

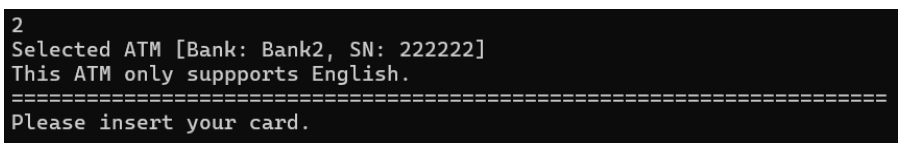
8. Multi-language support

▼(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.



[REQ8.1.1] Bilangual ATM에서 preferred language option 선택 화면

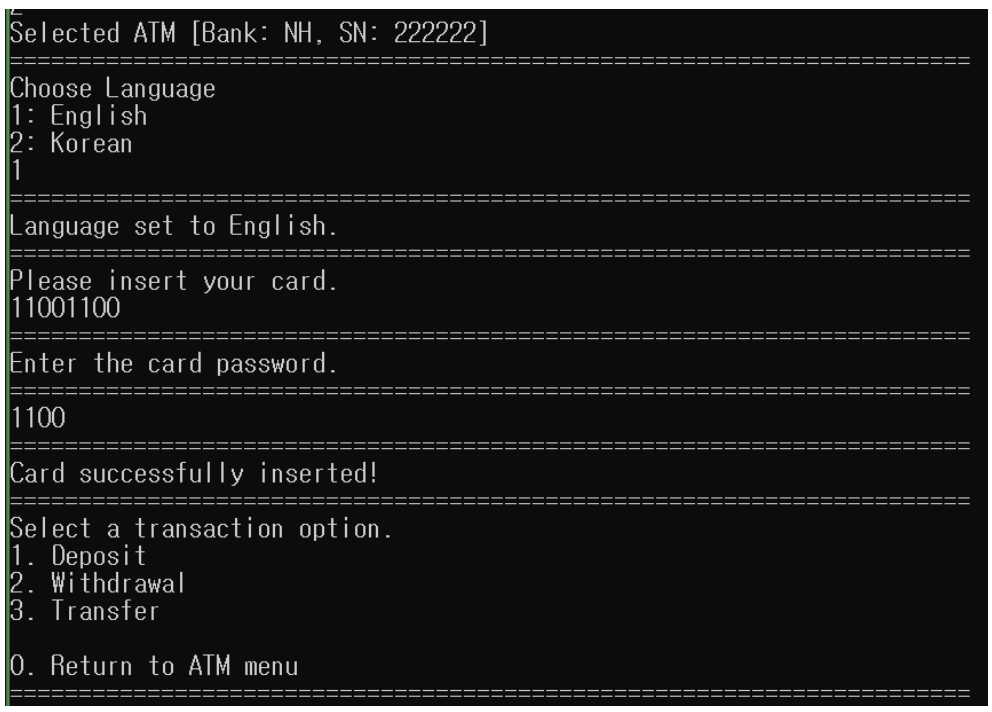
- ATM 2는 Bilangual ATM이기 때문에 영어 혹은 한국어를 선택할 수 있는 선택지가 나온다.



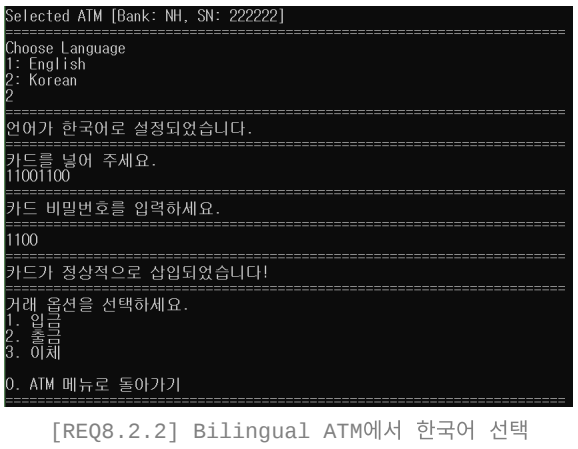
[REQ8.1.2] Bilangual ATM에서 preferred language option 선택 화면

- Unilangual ATM인 경우 영어 혹은 한국어 선택지 없이 영어로만 사용할 수 있다고 안내한다.

▼(REQ8.2) Once a certain language is chosen, all menus must be displayed using the chosen language.



[REQ8.2.1] Bilingual ATM에서 English 선택



[REQ8.2.2] Bilingual ATM에서 한국어 선택

- 위는 영어를 선택한 경우이고 아래는 한국어를 선택한 경우로, 선택지에서 영어 또는 한국어가 선택된다면 이후 화면에는 선택된 언어로 ATM을 사용할 수 있다.

9. Exception Handling

▼(REQ9.1) The ATM shall display an appropriate error message for each exception scenario(both explicitly stated in this document and implicitly assumed ones), then end the session.

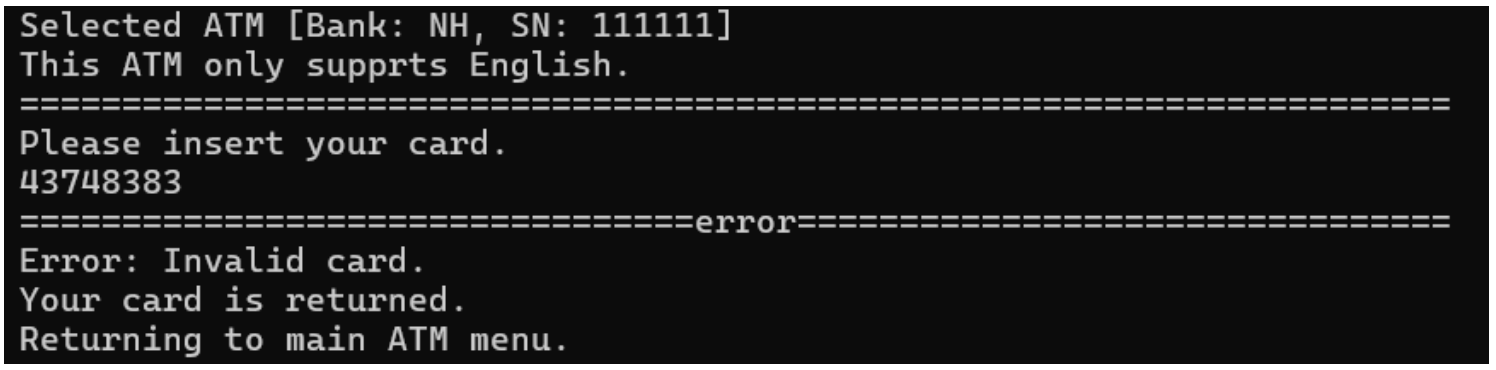
- (exception 1: WrongPassword)

  Card validation 과정에서 비밀번호를 3회 연속 틀리는 경우 발생하는 오류이다.

[REQ9.1.1] card validation 과정에서 비밀번호를 3회 연속 틀린 경우

- (exception 2: InvalidCard)

  ValidateCard 과정에서 존재하지 않는 카드를 입력한 경우 발생하는 오류이다.



[REQ9.2.1] card validation 과정에서 존재하지 않는 카드를 입력한 경우

- (exception 3: WrongInput)

  잘못된 선택지를 선택한 경우 발생하는 오류이다.

```
================================ Main ================================
Please select a task option.
1: Bank (Create or Use Bank)
2: User (Create or Use User)
3: ATM (Create or Use ATM)
/: Display all information

0: Exit Program
======================================================================
a
Invalid Input. Please try again.
```

[REQ9.3.1] Main 화면에서 잘못된 input(a) 입력한 경우

```
======================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
======================================================================
8
================================error=================================
오류: 잘못된 입력입니다.
======================================================================
```

[REQ9.3.2] Transaction option 화면에서 잘못된 input(8) 입력한 경우

- (exception 4: OverDeposit)

  deposit 한도를 넘어설 경우 발생하는 오류이다.

```
======================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
======================================================================
1
======================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 1
Enter number of KRW 10000: 2
Enter number of KRW 5000: 3
Enter number of KRW 1000: 50
======================================================================
================================error=================================
Error: Deposit limit exceeded.
======================================================================
The session ended.
======================================================================
```

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
2
==========================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
==========================================================================
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000

100000
100000
100000
100000
100000
100000
100000
100000
100000
100000

100000
==================================error=================================
Error: Deposit limit exceeded.
==========================================================================
The session ended.
==========================================================================
```

[REQ9.4.1] check deposit에서 30장을 초과하는 check를 deposit하려 한 경우

```
==========================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
==========================================================================
1
==========================================================================
Please put in as many bills as you want.

Enter number of KRW 50000: 1
Enter number of KRW 10000: 2
Enter number of KRW 5000: 3
Enter number of KRW 1000: 50
==========================================================================
==================================error=====================================
Error: Deposit limit exceeded.
==========================================================================
The session ended.
==========================================================================
```

[REQ9.4.2] cash deposit에서 각 지폐의 합이 50장을 초과하는 cash deposit을 시행하려 한 경우

- (exception 5: InsufficientFunds)

  withdrawal 또는 accout transfer에서 source account에 거래를 수행하기 위한 충분한 거래 잔고가 없을 경우 발생하는 오류이다.

```
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
200000
==================================error=====================================
Error: Insufficient account balance.
==========================================================================
The session ended.
==========================================================================
```

[REQ9.5.1] withdrawal에서 account에 충분한 잔고가 없는 경우

- (exception 6: ATMInsufficientFunds)

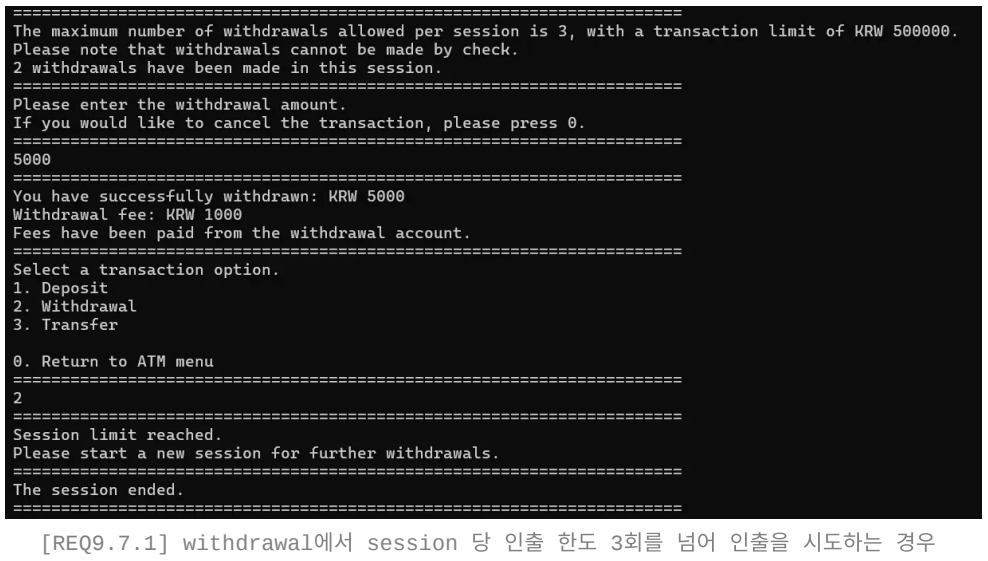
  withdrawal 과정에서 ATM에 인출할 만큼 충분한 양의 지폐가 존재하지 않는 경우 발생하는 오류이다.

```
==========================================================================
The maximum number of withdrawals allowed per session is 3, with a transaction limit of
 KRW 500000.
Please note that withdrawals cannot be made by check.
0 withdrawals have been made in this session.
==========================================================================
Please enter the withdrawal amount.
If you would like to cancel the transaction, please press 0.
==========================================================================
50000
==================================error=====================================
Error: Insufficient ATM balance.
==========================================================================
The session ended.
==========================================================================
```

[REQ9.6.1] withdrawal에서 ATM에 충분한 지폐 잔고가 없는 경우

- (exception 7: OverWithdrawPerSession)

  withdrawal에서 한 session 당 인출 한도인 3회를 넘어 인출을 시도하는 경우 발생하는 오류이다.

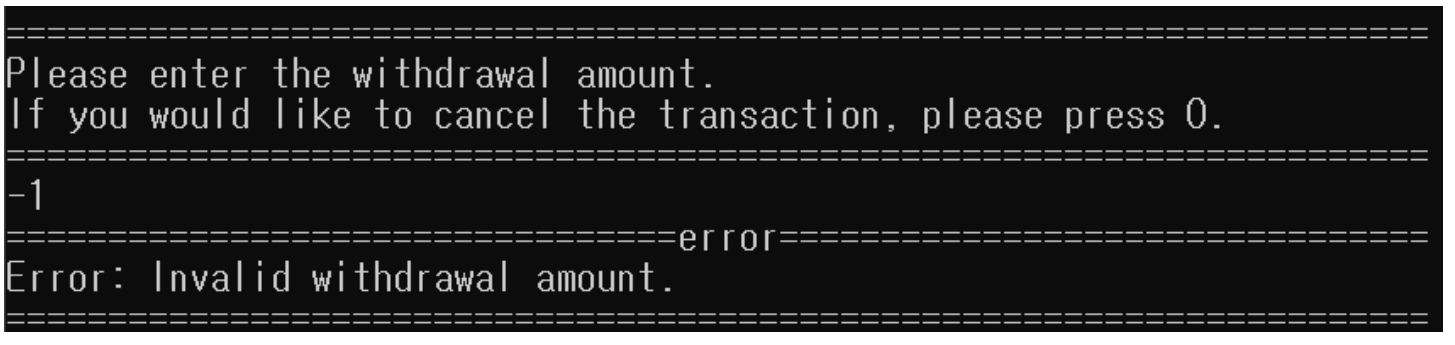


[REQ9.7.1] withdrawal에서 session 당 인출 한도 3회를 넘어 인출을 시도하는 경우

해당 콘솔은 withdrawal을 2회 시행한 이후, 3회차와 4회차의 withdrawal을 시행하는 상황을 보여준다. 이때 4회차에서는 withdrawal이 시행되지 않고 OverWithdrawalPerSession이 발생하는 것을 확인할 수 있다.

- (exception 8: InvalidWithdrawal)

  인출 금액이 0원 이하인 경우 발생하는 오류이다.



[REQ9.8.1] Withdrawal 금액이 0원 이하인 경우

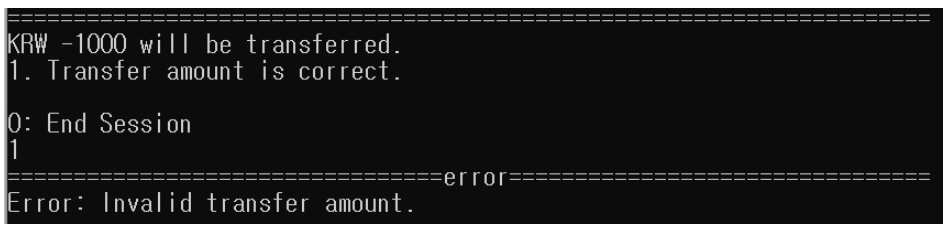
- (exception 9: OverWithdrawal)

  withdrawal에서 인출 금액이 500000원 초과인 경우 발생하는 오류이다.

[REQ9.9.1] Withdrawal 금액이 500000원 초과인 경우

- (exception 10: InvalidTransfer)

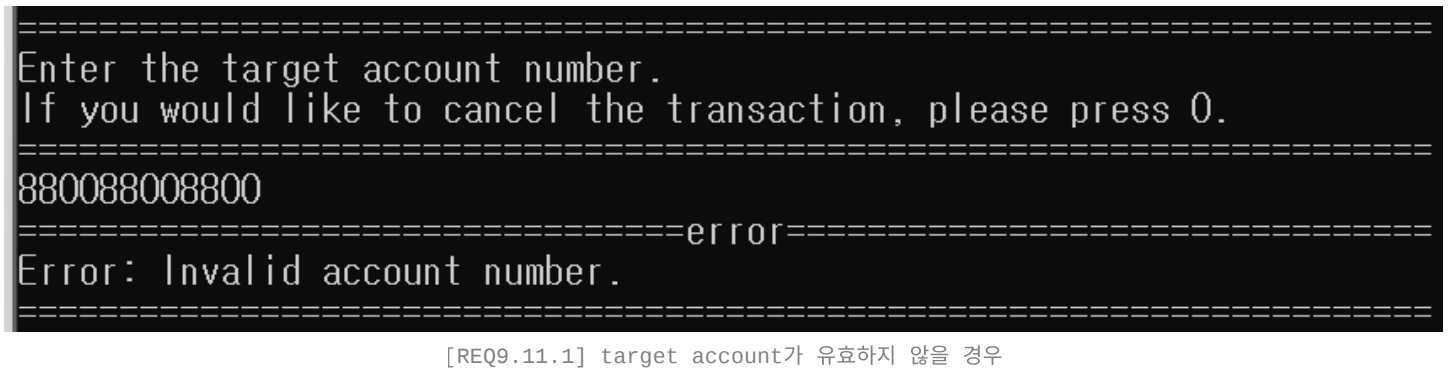
  cash transfer에서 이체 금액에 수수료를 뺀 금액이 0원 이하일 경우 발생하는 오류이다.



[REQ9.10.1] cash transfer에서 이체 금액에 수수료를 뺀 금액이 0원 이하일 경우

- (exception 11: InvalidAccount)

  transfer에서 target account가 유효하지 않을 경우 발생하는 오류이다.



[REQ9.11.1] target account가 유효하지 않을 경우

- (exception 12: FileError)

  admin inferface에서 transaction history 보기를 선택하였을 때 함께 진행되는 거래 내역 파일 저장에서 모종의 이유로 파일이 생성되고 열리지 않을 경우 발생하는 오류이다.

```
void ATM::TransactionHistorytoFile() {
    ofstream file("transaction_history.txt");
    if (file.is_open()) {
        for (size_t i = 0; i < ATM_history.size(); i++) {
            for (size_t j = 0; j < ATM_history[i].size(); j++) {
                file << ATM_history[i][j] << endl;
            }
        }
        file.close();
        if (language_option == 1) { cout << "Transaction history saved to transaction_history.txt" << endl; }
        else if (language_option == 2) { cout << "거래 내역이 transaction_history.txt에 저장되었습니다." << endl; }
    }
    else {
        ShowError(FileError);
        return;
    }
}
```

[REQ9.12.1] ATM의 TransactionHistorytoFile 함수

  FileError 관련 코드

  예상치 못한 파일 생성 및 열기 오류가 생겼을 때 FileError가 시행된다. 오류: 파일을 열 수 없습니다. 혹은 Error: Unable to open file. 라고 한국어, 영어가 language type에 맞게 각각 출력된다.

- (exception 13: UnusableCheck)

  ATM에서 취급하지 않는 100000원 미만 금액의 Check를 받은 경우 발생하는 오류이다.

```
================================================================================
Which one do you want to deposit, cash or check?
1. Cash
2. Check

0. Cancel
================================================================================
2
================================================================================
Please enter the check up to 30 papers.
If you want to stop entering the check, press 0.
================================================================================
4000
=================================error==========================================
Error: Unusable Check. Only over 100000 won of check can be used.
================================================================================
The session ended.
================================================================================
```

[REQ9.13.1] 100000 미만의 Check 금액을 입력한 경우

- (exception 14: UnusableCard)

  single Bank ATM에서 타 은행의 카드를 삽입한 경우 발생하는 오류이다.

```
================================================================================
Please insert your card.
11001100
=================================error==========================================
Error: Unusable Card for this ATM.
================================================================================
```

[REQ9.14.1] card validation 과정에서 Single Bank ATM에 타 은행의 카드를 입력했을 경우

  ATM 4은 우리 은행의 Single Bank ATM이나, card는 NH 은행의 것으로 은행이 일치하지 않아 InvalidCard 오류가 발생한다.

10. Display of Account/ATM Snapshot

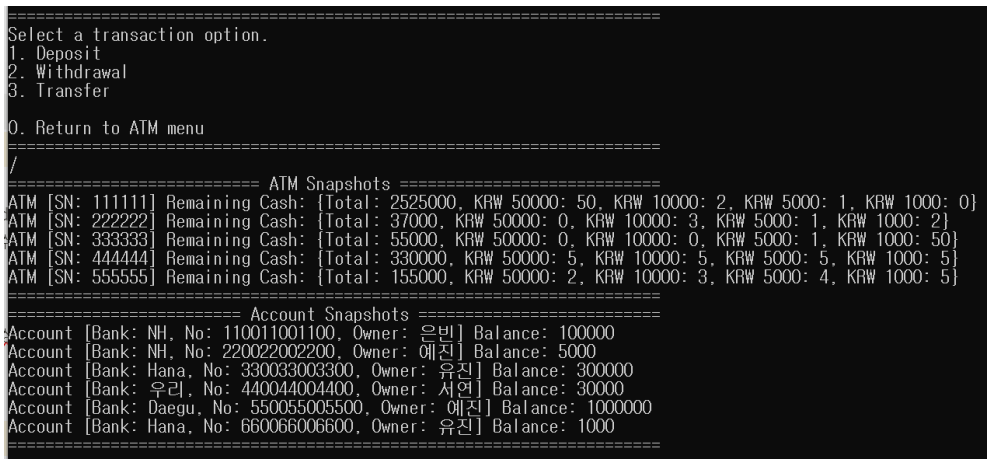
▼(REQ10.1) When the character ' / ' (slash) is given as a console input during the program execution, the following information shall be displayed to the console.

10.1.(1) All ATMs’ information: Remaining cash

(e.g., ATM [SN: 111111] remaining cash: {KRW 50000 : 0, KRW 10000 : 1, KRW 5000 : 2, KRW 1000 : 1}, ATM [SN: 222222] remaining cash: {KRW 50000 : 5, KRW 10000 : 3, KRW 5000 : 1, KRW 1000 : 2})

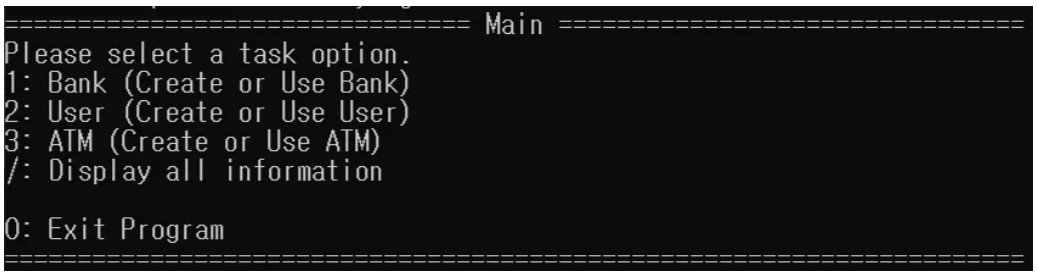
10.1.(2) All accounts’ information: Remaining balance

(e.g., Account [Bank: Kakao, No: 111111111111, Owner: Jenny] balance: 7000, Account [Bank: Daegu, No: 222222222222, Owner: Tom] balance: 1000, Account [Bank: Shinhan, No: 333333333333, Owner: Jenny] balance: 2000)

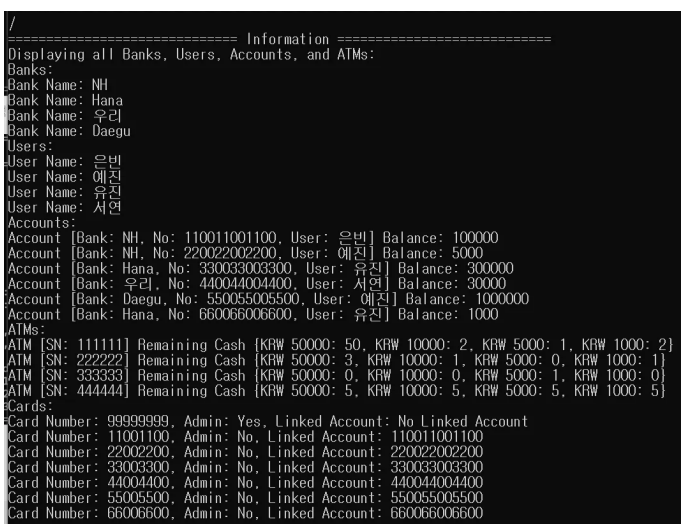


[REQ10.1.1] session 내 거래 선택 창에서 snapshot을 출력한 모습

- program 실행 중 session 내 거래 선택 창에서 ‘/’를 콘솔에 입력한 경우, ATM Snapshot과 Account Snapshot이 구분되어 출력된다.
- ATM Snapshot에는 일련번호, 남은 지폐를 합계 금액, 50000원 권, 10000원 권, 5000원 권, 1000원 권 개수 순으로 출력된다.
- Account Snapshot에는 은행, 계좌번호, 소유자, 잔고 순으로 출력된다.



[REQ10.1.2]main에서 Display all information(/) 선택 가능



[REQ10.1.3] Display all information(/) 입력 시 출력되는 정보

- program 실행 중 initialization시 ‘/’를 콘솔에 입력한 경우, Bank, User, Account, ATM, Card 내역이 순차적으로 출력된다.
- ATM에는 일련번호, 남은 지폐의 개수가 50000원, 10000원, 5000원, 1000원 순으로 출력된다.
- Account에는 은행, 계좌번호, 소유자, 잔고 순으로 출력된다.
- 이외에도 bank에는 모든 bank의 종류, user에는 모든 user의 종류, card에는 카드번호, admin 여부, 연결된 계좌 내역이 출력된다.

5. Concepts of OOP

1. Inheritance

- Deposit, Withdrawal, Transfer class 구현 시 Transaction class를 상속받아 만들었기 때문에, transaction의 protected 내에 존재하는 member를 Deposit, Withdrawal, Transfer 내에서 사용할 수 있다.
  - SetFee()의 경우는 virtual 함수로 설정해 각 child class마다 다른 수수료를 설정할 수 있게 하였다.
- Admincard class의 경우에도 Card class를 상속받아 만들었기 때문에 Card class의 member를 상속받았다.

▼ (관련 코드)

```
// ================================== Transaction ==================================

class Transaction {
protected:
    string transaction_type;
    ATM* atm;
    int transaction_fee;
    int one_thousand, five_thousand, ten_thousand, fifty_thousand;
    int transaction_amount;
    string additional_info;
    Card* inserted_card;
    Account* source_account;
    Account* target_account;
public:
    Transaction(ATM* atm, Card* inserted_card) : atm(atm), inserted_card(inserted_card) {}
    virtual void SetFee() = 0;
    virtual void GetDeficientFee(Card*);
    string GetTransactionName() { return transaction_type; }
    int GetCardNumber() { return inserted_card->GetCardNumber(); }
    void SetTransactionAmount() { transaction_amount = one_thousand * 1000 + five_thousand * 5000 + ten_thousand * 10000 + fifty_thousand * 50000; }
    int GetTransactionAmount() { return transaction_amount; }
    Account* GetSourceAccount() { return source_account; }
    Account* GetTargetAccount() { return target_account; }
};



 // ================================== Deposit ==================================w

 class Deposit : public Transaction {
 private:
     int choose_cash;
     int num_of_check;
     long long int sum_of_check = 0;
     long long int incheck;
 public:
     Deposit(ATM* d_atm, Card* d_inserted_card) : Transaction(d_atm, d_inserted_card) { transaction_type = "Deposit"; }
     void SetFee();
     void GetDeficientFee();
     void DepositInprogress();
     void GetCash(int one_thousand, int five_thousand, int ten_thousand, int fifty_thousand);
     void GetCheck();
 };

 // ================================== Withdrawal ==================================

class Withdrawal : public Transaction {
private:
    const int max_drawal_cash = 500000;
    const int max_drawals_session = 3;
public:
    static int withdrawal_count;
    Withdrawal(ATM* w_atm, Card* w_inserted_card) : Transaction(w_atm, w_inserted_card) { transaction_type = "Withdrawal"; }
    void SetFee();
    void WithdrawalInprogress();
};



// ================================== Transfer ==================================

class Transfer : public Transaction {
private:
    int final_transfer_amount;
    int inserted_amount;
    int final_minus_amount;
    int transfer_type;
    int input_transaction_fee;

public:
    Transfer(ATM* t_atm, Card* t_inserted_card) : Transaction(t_atm, t_inserted_card) { transaction_type = "Transfer"; }
    void GetDeficientFee();
    void SetFee();
    void TransferInprogress();
    void CashTransfer();
    void AccountTransfer();
};



          // ================================== Card ==================================

          class Card {
          private:
              int card_number;         // 8-digit
              bool admin = false;
              int card_password;
              Account* linked_account;
          protected:
              static vector<Card*> card_list;
          public:
              Card(Account* c_linked_account);
              Card();
              void SetCardPassword() { card_password = GetPassword(); }
              bool GetAdminBool() { return admin; }
              string GetBankName() { return linked_account->GetBankName(); }
              int GetCardNumber() { return card_number; }
              long long int GetAccountNumber() { return linked_account->GetAccountNumber(); }
              int GetPassword() { return linked_account->GetPassword(); }
              void SetAdminBool(bool c_admin) { admin = c_admin; }
              Account* GetAccount() { return linked_account; }
              friend Card* FindCard(int c_card_number);
              void SetCardNumber(int c_card_number) { card_number = c_card_number; }
              static vector<Card*> GetCardList() { return card_list; }
              string ToString() {
                  string linked_account_info = linked_account ? to_string(linked_account->GetAccountNumber()) : "No Linked Account";
                  return "Card Number: " + to_string(card_number) +
                      ", Admin: " + ((admin == true) ? "Yes" : "No") +
                      ", Linked Account: " + linked_account_info;
              }
          };



// ================================== Admin Card ==================================

class AdminCard : public Card {
public:
    AdminCard() : Card() {
        Card::SetCardNumber(99999999);
        SetAdminBool(true);
        card_list.push_back(this);
        cout << "AdminCard created with Card Number: 99999999" << endl;
    }
};
```

위쪽부터 Card, Admincard class의 코드이다.

2. **Polymorphism**

- Transaction class의 virtual 함수와 이를 override한 Deposit, Withdrawal, Transfer 각각의 class에서 polymorphism을 발견할 수 있다.
- SetFee()를 예시로 들면, parent class의 virtual 함수 SetFee()에 대한 서술은 각각의 child class에서 이루어진다.

▼ (관련 코드)

```
// ================================== Transaction ==================================

class Transaction {
protected:
    string transaction_type;
    ATM* atm;
    int transaction_fee;
    int one_thousand, five_thousand, ten_thousand, fifty_thousand;
    int transaction_amount;
    string additional_info;
    Card* inserted_card;
    Account* source_account;
    Account* target_account;
public:
    Transaction(ATM* atm, Card* inserted_card) : atm(atm), inserted_card(inserted_card) {}
    virtual void SetFee() = 0;
    virtual void GetDeficientFee(Card*);
    string GetTransactionName() { return transaction_type; }
    int GetCardNumber() { return inserted_card->GetCardNumber(); }
    void SetTransactionAmount() { transaction_amount = one_thousand * 1000 + five_thousand * 5000 + ten_thousand * 10000 + fifty_thousand * 50000; }
    int GetTransactionAmount() { return transaction_amount; }
    Account* GetSourceAccount() { return source_account; }
    Account* GetTargetAccount() { return target_account; }
};


// ========================== Deposit External Declaration ==========================

void Deposit::SetFee() {
    if (target_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else { transaction_fee = 2000; }
}
```

—> withdrawal SetFee()

```
void Withdrawal::SetFee() {
    if (source_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else {
        transaction_fee = 2000;
    }
}
```

```
// ========================== Transfer External Declaration ==========================

void Transfer::SetFee() {
    if (transfer_type == 2) {
        if (source_account->GetBankName() == atm->GetPrimaryBankName() && target_account->GetBankName() == atm->GetPrimaryBankName()) {
            transaction_fee = 2000;
        }
        else if (source_account->GetBankName() != atm->GetPrimaryBankName() && target_account->GetBankName() != atm->GetPrimaryBankName()) {
            transaction_fee = 4000;
        }
        else {
            transaction_fee = 3000;
        }
    }
    else if (transfer_type == 1) {
        transaction_fee = 1000;
    }
}
```

위쪽부터 Deposit, Withdrawal, Transfer의 SetFee 함수이다. 각각의 함수 내용에 차이가 있음을 확인할 수 있다.

3. **Encapsulation**

- encapsulation을 보장하기 위해 다양한 class를 만들어 private에 속한 member는 다른 함수에서 접근할 수 없고, protected에 속한 member는 그 class의 child class에서만 접근할 수 있게 하였다.

▼(관련 코드)

```
// ============================ Class Forward Declaration ============================

class User;
class Account;
class Bank;
class Card;
class AdminCard;
class Transaction;
class Deposit;
class Withdrawal;
class Transfer;
class ATM;
```

다양한 class들.

```
class Card {
private:
    int card_number;         // 8-digit
    bool admin = false;
    int card_password;
    Account* linked_account;
protected:
    static vector<Card*> card_list;
public:
    Card(Account* c_linked_account);
    Card();
    void SetCardPassword() { card_password = GetPassword(); }
    bool GetAdminBool() { return admin; }
    string GetBankName() { return linked_account->GetBankName(); }
    int GetCardNumber() { return card_number; }
    long long int GetAccountNumber() { return linked_account->GetAccountNumber(); }
    int GetPassword() { return linked_account->GetPassword(); }
    void SetAdminBool(bool c_admin) { admin = c_admin; }
    Account* GetAccount() { return linked_account; }
    friend Card* FindCard(int c_card_number);
    void SetCardNumber(int c_card_number) { card_number = c_card_number; }
    static vector<Card*> GetCardList() { return card_list; }
    string ToString() {
        string linked_account_info = linked_account ? to_string(linked_account->GetAccountNumber()) : "No Linked Account";
        return "Card Number: " + to_string(card_number) +
            ", Admin: " + ((admin == true) ? "Yes" : "No") +
            ", Linked Account: " + linked_account_info;
    }
};
```

예시. Card class. 내부에 private 및 protected에 attribute를 할당함으로써 외부의 접근을 막는 encapsulation이 적용되었다.

4. **Abstraction**

- 사용자에게는 우리가 cout으로 출력하도록 설정한 부분만 보여주고, 내부 설정은 class에서 내부적으로 실행될 수 있게 하여 abstraction을 적용하였다.
- transaction에서 virtual 함수를 정의하여 child class의 공통된 특성을 추상화하였다.

▼(관련 코드)

```
void Withdrawal::WithdrawalInprogress() {
    while (true) {
        if (withdrawal_count == 0) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500000.\nPlease note that withdrawals cannot be made by check." << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "한 세션당 최대 3회의 출금이 가능하며, 거래 한도는 500000원입니다.\n수표로는 출금이 불가능하니 유의해주십시오." << endl;
            }
        }
        else if (withdrawal_count == 3) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Session limit reached.\nPlease start a new session for further withdrawals." << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "세션당 출금은 최대 3회만 가능합니다.\n추가 출금을 원하신다면, 새로운 세션을 시작해주세요." << endl;
            }
            atm->SessionEnds();
            break;
        }
        else {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "The transaction limit is KRW 500000.\nPlease note that withdrawals cannot be made by check." << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "거래 한도는 500000원입니다. 수표로는 출금이 불가능하니 유의해주십시오." << endl;
            }
        }
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Please enter the withdrawal amount." << endl;
            cout << "If you would like to cancel the transaction, please press 0." << endl;
            cout << "================================================================================" << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "출금하실 금액을 입력해주세요." << endl;
            cout << "거래 취소를 원하신다면, 0을 입력해주세요." << endl;
            cout << "================================================================================" << endl;
        }
        cin >> transaction_amount;
        if (transaction_amount == 0) {
            atm->SessionEnds();
        }

        Bank* w_source_bank = FindBank(inserted_card->GetBankName());
        source_account = w_source_bank->ValidateCard(inserted_card);
        if (source_account == nullptr) {
            atm->ShowError(InvalidCard);
        }
        SetFee();

        if (transaction_amount <= 0) {
            atm->ShowError(InvalidWithdrawal);
            return;
        }

        if (transaction_amount > max_drawal_cash) {
            atm->ShowError(OverWithdrawal);
            return;
        }

        if (source_account->GetAvailableFund() < (transaction_amount + transaction_fee)) {
            atm->ShowError(InsufficientFunds);
            return;
        }
        int ATM_fifty_thousand = atm->GetFiftyThousand();
        int ATM_ten_thousand = atm->GetTenThousand();
        int ATM_five_thousand = atm->GetFiveThousand();
        int ATM_one_thousand = atm->GetOneThousand();

        int remaining_amount = transaction_amount;
        fifty_thousand = min(remaining_amount / 50000, ATM_fifty_thousand);
        remaining_amount -= fifty_thousand * 50000;
        ten_thousand = min(remaining_amount / 10000, ATM_ten_thousand);
        remaining_amount -= ten_thousand * 10000;


                five_thousand = min(remaining_amount / 5000, ATM_five_thousand);
                remaining_amount -= five_thousand * 5000;
                one_thousand = min(remaining_amount / 1000, ATM_one_thousand);
                remaining_amount -= one_thousand * 1000;

                int fifty_thousand2;
                int ten_thousand2;
                int five_thousand2;
                int one_thousand2;

                int remaining_amount2 = transaction_amount;
                fifty_thousand2 = remaining_amount2 / 50000;
                remaining_amount2 %= 50000;
                ten_thousand2 = remaining_amount2 / 10000;
                remaining_amount2 %= 10000;
                five_thousand2 = remaining_amount2 / 5000;
                remaining_amount2 %= 5000;
                one_thousand2 = remaining_amount2 / 1000;
                remaining_amount2 %= 1000;

                if (remaining_amount2 > 0) {
                    atm->ShowError(WrongInput);
                    return;
                }
                if (remaining_amount > 0) {
                    atm->ShowError(ATMInsufficientFunds);
                    return;
                }
                if (HasATMEnoughMoney() == false) {
                    return;
                }
                if (ATM::GetLanguageOption() == 1) {
                    cout << "================================================================================" << endl;
                    cout << "You have successfully withdrawn: KRW " << transaction_amount << endl;
                    cout << "Transaction fee: " << transaction_fee << " KRW" << endl;
                    cout << "Fees have been paid from the withdrawal account." << endl;
                }
                else {
                    cout << "================================================================================" << endl;
                    cout << "성공적으로 출금되었습니다: " << transaction_amount << " 원" << endl;
```

```
            cout << "거래 수수료: " << transaction_fee << " 원" << endl;
            cout << "수수료가 출금 계좌에서 차감되었습니다." << endl;
        }

        source_account->UpdateAccountMoney(transaction_amount + transaction_fee, 0);
        atm->UpdateATMFund(one_thousand, five_thousand, ten_thousand, fifty_thousand, 0);
        withdrawal_count++;

        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================" << endl;
            cout << "Would you like to make another withdrawal?\n1. yes\n2. no\n";
        }
        else {
            cout << "================================================================" << endl;
            cout << "추가 출금을 원하십니까?\n1. 예\n2. 아니오\n";
        }

        int more_withdrawal_choice;
        cin >> more_withdrawal_choice;
        if (more_withdrawal_choice == 2) {
            atm->AddATMHistory(this);
            break;
        }
        else if (more_withdrawal_choice == 1) {
            atm->AddATMHistory(this);
            continue;
        }
        else {
            atm->ShowError(WrongInput);
            return;
        }
    }
}
```

예시로 WithdrawalInprogress에서, 사용자는 cout으로 출력되는 부분만을 볼 수 있지만, 프로그램 내에서는 인출 횟수 세기, 거래 한도 확인, account의 유효 확인, 계좌 잔고와 ATM의 지폐 종류별 개수 확인, 계좌 잔액 및 ATM 지폐 종류별 개수 업데이트, 거래 기록 등의 활동이 사용자가 알지 못하는 사이 실행된다.

```
// ================================== Transaction ==================================

class Transaction {
protected:
    string transaction_type;
    ATM* atm;
    int transaction_fee;
    int one_thousand, five_thousand, ten_thousand, fifty_thousand;
    int transaction_amount;
    string additional_info;
    Card* inserted_card;
    Account* source_account;
    Account* target_account;
public:
    Transaction(ATM* atm, Card* inserted_card) : atm(atm), inserted_card(inserted_card) {}
    virtual void SetFee() = 0;
    virtual void GetDeficientFee(Card*);
    string GetTransactionName() { return transaction_type; }
    int GetCardNumber() { return inserted_card->GetCardNumber(); }
    void SetTransactionAmount() { transaction_amount = one_thousand * 1000 + five_thousand * 5000 + ten_thousand * 10000 + fifty_thousand * 50000; }
    int GetTransactionAmount() { return transaction_amount; }
    Account* GetSourceAccount() { return source_account; }
    Account* GetTargetAccount() { return target_account; }
};


// ================================== Deposit ==================================w

class Deposit : public Transaction {
private:
    int choose_cash;
    int num_of_check;
    long long int sum_of_check = 0;
    long long int incheck;
public:
    Deposit(ATM* d_atm, Card* d_inserted_card) : Transaction(d_atm, d_inserted_card) { transaction_type = "Deposit"; }
    void SetFee();
    void GetDeficientFee();
    void DepositInprogress();
    void GetCash(int one_thousand, int five_thousand, int ten_thousand, int fifty_thousand);
    void GetCheck();
};

// ================================== Withdrawal ==================================

class Withdrawal : public Transaction {
private:
    const int max_drawal_cash = 500000;
    const int max_drawals_session = 3;
public:
    static int withdrawal_count;
    Withdrawal(ATM* w_atm, Card* w_inserted_card) : Transaction(w_atm, w_inserted_card) { transaction_type = "Withdrawal"; }
    void SetFee();
    void WithdrawalInprogress();
};


// ================================== Transfer ==================================

class Transfer : public Transaction {
private:
    int final_transfer_amount;
    int inserted_amount;
    int final_minus_amount;
    int transfer_type;
    int input_transaction_fee;

public:
    Transfer(ATM* t_atm, Card* t_inserted_card) : Transaction(t_atm, t_inserted_card) { transaction_type = "Transfer"; }
    void GetDeficientFee();
    void SetFee();
    void TransferInprogress();
    void CashTransfer();
    void AccountTransfer();
};
```

위쪽부터 transaction, deposit, withdrawal, transfer class의 declaration이다. virtual 함수를 통한 추상화가 적용되어 있다.

5. **Exception handling**

- ATM에서 발생하지 않아야 할 몇 가지 예외 사항의 경우에 ShowError()이라는 함수를 이용하여 예외 처리를 하였다.
- 알 수 없는 예외 상황의 경우에는 함수의 default 처리를 하여 unknown Error이라는 문구를 띄울 수 있게 하였다.
- C++의 대표적인 exception handling 기법인 try-throw-catch 구문을 이용하여 card 생성 중 카드 번호가 8자리가 아닐 경우 card 생성을 중지하고 Error를 출력하게 하였다.

▼ (관련 코드)

```
// ============================== Error Type Declaration ==============================

enum ErrorType {
    WrongPassword,
    InvalidCard,
    WrongInput,
    OverDeposit,
    InsufficientFunds,
    ATMInsufficientFunds,
    OverWithdrawPerSession,
    InvalidWithdrawal,
    OverWithdrawal,
    InvalidTransfer,
    InvalidAccount,
    FileError,
    UnusableCheck,
    UnusableCard
};


void ATM::ShowError(ErrorType ShowErrortype) {
    cout << "================================error================================" << endl;
    if (language_option == 1) {
        switch (ShowErrortype) {
        case WrongPassword: cout << "Error: Incorrect password." << endl; break;
        case InvalidCard: cout << "Error: Invalid card." << endl; break;
        case InvalidAccount: cout << "Error: Invalid account number." << endl; break;
        case WrongInput: cout << "Error: Invalid input." << endl; break;
        case OverDeposit: cout << "Error: Deposit limit exceeded." << endl; break;
        case InsufficientFunds: cout << "Error: Insufficient account balance." << endl; break;
        case ATMInsufficientFunds: cout << "Error: Insufficient ATM balance." << endl; break;
        case OverWithdrawPerSession: cout << "Error: Withdrawal limit for this session reached. Please start a new session for additional withdrawals." << endl; break;
        case InvalidWithdrawal: cout << "Error: Invalid withdrawal amount." << endl; break;
        case OverWithdrawal: cout << "Error: Single deposit limit of 500000 won exceeded." << endl; break;
        case InvalidTransfer: cout << "Error: Invalid transfer amount." << endl; break;
        case FileError: cout << "Error: Unable to open file." << endl; break;
        case UnusableCheck: cout << "Error: Unusable Check. Only over 100000 won of check can be used." << endl; break;
        case UnusableCard: cout << "Error: Unusable Card for this ATM." << endl; break;
        default: cout << "Error: Unknown Error" << endl;
        }
    }
    else if (language_option == 2) {
        switch (ShowErrortype) {
        case WrongPassword: cout << "오류: 잘못된 비밀번호입니다." << endl; break;
        case InvalidCard: cout << "오류: 유효하지 않은 카드입니다." << endl; break;
        case InvalidAccount: cout << "오류: 유효하지 않은 계좌번호입니다." << endl; break;
        case WrongInput: cout << "오류: 잘못된 입력입니다." << endl; break;
        case OverDeposit: cout << "오류: 예치 한도가 초과되었습니다." << endl; break;
        case InsufficientFunds: cout << "오류: 계좌 잔액이 부족합니다." << endl; break;
        case ATMInsufficientFunds: cout << "오류: ATM 잔액이 부족합니다." << endl; break;
        case OverWithdrawPerSession: cout << "오류: 세션 당 출금 한도에 도달하였습니다. 추가 출금을 원하신다면 새로운 세션을 시작해주세요." << endl; break;
        case InvalidWithdrawal: cout << "오류: 올바르지 않은 출금 금액입니다. " << endl; break;
        case OverWithdrawal: cout << "오류: 1회 예치 한도 500000원을 초과하였습니다. " << endl; break;
        case InvalidTransfer: cout << "오류: 올바르지 않은 이체 금액입니다." << endl; break;
        case FileError: cout << "오류: 파일을 열 수 없습니다." << endl; break;
        case UnusableCheck: cout << "오류: 사용할 수 없는 수표입니다. 100000원 이상의 수표만 사용 가능합니다." << endl; break;
        case UnusableCard: cout << "오류: 해당 ATM에서 사용할 수 없는 카드입니다." << endl; break;
        default: cout << "오류: 알 수 없는 오류입니다." << endl;
        }
    }
```

ShowError 함수 이용 예외 처리

```
// ========================== Account External Declaration ==========================

Account::Account(Bank* bank_ptr, string a_user_name, long long int acc_num, int funds, int pass)
    : bank(bank_ptr), account_number(acc_num), available_fund(funds), password(pass) {
    user = FindUser(a_user_name);
    if (!user) {
        cout << "User does not exist. Please create the user first." << endl;
        return;
    }

    try {
        Card* new_card = new Card(this);
        if (new_card) {
            cout << "Card created successfully for Account: " << this->GetAccountNumber() << endl;
        }
    }
    catch (const invalid_argument& e) {
        cout << "Account creation aborted due to invalid card number. Please try again." << endl;
        throw;
    }
    account_list.push_back(this);
}
```

```
Card::Card(Account* c_linked_account) {
    if (!admin) {
        cout << "=====================================================================" << endl;
        cout << "Card Number: ";
        int c_card_number;
        cin >> c_card_number;
        if (to_string(c_card_number).length() != 8) {
            cout << "Invalid Card Number. Please enter an 8-digit number." << endl;
            cout << "=====================================================================" << endl;
            throw invalid_argument("Invalid Card Number");
        }
        card_number = c_card_number;

        linked_account = c_linked_account;
        if (!linked_account) {
            cout << "Error: Linked account is null!" << endl;
            return;
        }

        SetCardPassword();
        cout << "=====================================================================" << endl;
        card_list.push_back(this);
    }
}
```

try-throw-catch 구문 이용 예외 처리

6. STL

- vector를 이용하여 User class의 user_list, Bank class의 bank_list, Account class의 account_list, Card class의 card_list, ATM class의 ATM_history와 atm_list 등을 저장하였다.
- user_list, bank_list, account_list, card_list, atm_list는 초기 initialization의 상황에서 입력 받은 user, bank, account, card, atm을 각각 vector의 형태로 저장한다.
- ATM_history에서는 거래 기록을 추후 출력하기 위해 vector의 형태로 각 거래별 기록을 저장한다. vector에 기본적으로 내장되어 있는 push_back(), size() 등의 member methods를 이용하여 기록 및 출력을 진행한다.

▼ (관련 코드)

```
// ======================================== User ========================================

class User {
private:
    string user_name;
    static vector<User*> user_list;
public:
    User(string u_user_name);
    string GetUserName() { return user_name; }
    friend User* FindUser(string u_user_name);
    static vector<User*> GetUserList() { return user_list; }
};



// ======================================== Bank ========================================

class Bank {
private:
    string bank_name;
    static vector<Bank*> bank_list;
public:
    Bank(string b_bank_name);
    string GetBankName() { return bank_name; }
    static vector<Bank*> GetBankList() { return bank_list; }
    friend Bank* FindBank(string b_bank_name);
    Account* ValidateCard(Card* card);
    Account* ValidateAccount(long long int account_number);
};



            // ====================================== Account ======================================

            class Account {
            private:
                long long int account_number;       // 12-digit
                User* user;
                Bank* bank;
                int available_fund;
                int password;       // 4-digit
                static vector<Account*> account_list;
            public:
                Account(Bank* bank_ptr, string a_user_name, long long int acc_num, int funds, int pass);
                string GetBankName() { return bank->GetBankName(); }
                long long int GetAccountNumber() { return account_number; }
                int GetPassword() { return password; }
                int GetAvailableFund() { return available_fund; }
                void UpdateAccountMoney(int amount, int mark);
                string GetUserName() { return user ? user->GetUserName() : "Unknown"; }
                static vector<Account*> GetAccountList() { return account_list; }
                static Account* FindAccount(long long int account_number);
            };
```

```
// ======================================== Card ========================================

class Card {
private:
    int card_number;        // 8-digit
    bool admin = false;
    int card_password;
    Account* linked_account;
protected:
    static vector<Card*> card_list;
public:
    Card(Account* c_linked_account);
    Card();
    void SetCardPassword() { card_password = GetPassword(); }
    bool GetAdminBool() { return admin; }
    string GetBankName() { return linked_account->GetBankName(); }
    int GetCardNumber() { return card_number; }
    long long int GetAccountNumber() { return linked_account->GetAccountNumber(); }
    int GetPassword() { return linked_account->GetPassword(); }
    void SetAdminBool(bool c_admin) { admin = c_admin; }
    Account* GetAccount() { return linked_account; }
    friend Card* FindCard(int c_card_number);
    void SetCardNumber(int c_card_number) { card_number = c_card_number; }
    static vector<Card*> GetCardList() { return card_list; }
    string ToString() {
        string linked_account_info = linked_account ? to_string(linked_account->GetAccountNumber()) : "No Linked Account";
        return "Card Number: " + to_string(card_number) +
            ", Admin: " + ((admin == true) ? "Yes" : "No") +
            ", Linked Account: " + linked_account_info;
    }
};
```

위쪽부터 user, bank, account, card 클래스의 vector 사용을 확인할 수 있다.

```
// ======================================== ATM ========================================

class ATM {
private:
    int serial_number;         // 6-digit
    int ATM_type;         // single = 1, multi = 2
    Bank* primary_bank;
    int ATM_lan_type;         // unilingual = 1, bilingual = 2
    static int language_option;         // English = 1, Korean = 2
    int ATM_fund;
    int ATM_1000_num = 0, ATM_5000_num = 0, ATM_10000_num = 0, ATM_50000_num = 0;
    static int session_num;
    static int transaction_num;
    bool card_inserted = false;
    bool admin_access = false;
    vector<vector<string>> ATM_history;
    static vector<ATM*> atm_list;
    Deposit* deposit;
    Transfer* transfer;
    Withdrawal* withdrawal;


        // ============================== List Initialization ==============================

        vector<ATM*> ATM::atm_list = {};
        vector<Bank*> Bank::bank_list = {};
        vector<Account*> Account::account_list = {};
        vector<User*> User::user_list = {};
        vector<Card*> Card::card_list = {};
```

각 vector의 initialization 모습

```
void ATM::AddATMHistory(Transaction* transaction) {
    if (ATM_history.size() <= session_num) {
        ATM_history.resize(session_num + 1);
    }

    string history = ((language_option == 1) ? "Transaction ID: " : "거래 번호: ") + to_string(transaction_num) +
        ((language_option == 1) ? ", Card Number: " : ", 카드 번호: ") + to_string(transaction->GetCardNumber()) +
        ((language_option == 1) ? ", Transaction Type: " : ", 거래 유형: ") + transaction->GetTransactionName() +
        ((language_option == 1) ? ", Amount: KRW " : ", 금액: ") + to_string(transaction->GetTransactionAmount());

    if (transaction->GetTransactionName() == "Deposit") {
        history += ", Target Account Number: " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Withdrawal") {
        history += ", Source Account Number: " + to_string(transaction->GetSourceAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Cash Transfer") {
        history += ", Target Account Number : " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Account Transfer") {
        history += ", Source Account Number: " + to_string(transaction->GetSourceAccount()->GetAccountNumber()) +
            ", Target Account Number: " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }

    ATM_history[session_num].push_back(history);
    transaction_num++;
}
```

```
void ATM::ShowSessionHistory() {
    if (session_num >= ATM_history.size() || ATM_history[session_num].empty()) {
        if (language_option == 1) {
            cout << "No transaction history available for this session." << endl;
        }
        else if (language_option == 2) {
            cout << "해당 세션에서의 거래 내역이 없습니다." << endl;
        }
        return;
    }

    cout << "==========================================================================" << endl;
    cout << ((language_option == 1) ? "Session Transaction History" : "세션 거래 내역") << endl;
    for (const string& record : ATM_history[session_num]) {
        cout << record << endl;
    }
    cout << "==========================================================================" << endl;
}
```

ATM_history에서 vector의 member method를 이용하는 모습.

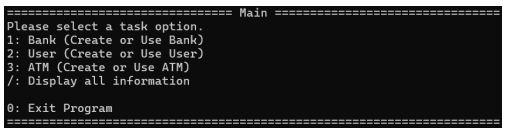
6. Instructions to Run the Source Code

0. 프로그램 실행

```
AdminCard created with Card Number: 99999999
```

- 실행과 동시에 Admin Card가 생성되었다.

1. Main



ⓐ Main 화면

먼저 ⓐ와 같이 수행할 task 선택지가 등장한다. 각 항목이 의미하는 바는 다음과 같다.

- 1: Bank와 관련된 task로, Bank를 생성하거나, 생성되어있는 Bank에 대하여 Account를 개설할 수 있다.
- 2: User와 관련된 task로, User를 새로 추가하거나, User 목록을 확인하고 선택한 User의 Account 목록을 확인할 수 있다.
- 3: ATM과 관련된 task로, ATM을 생성하거나, 생성된 ATM 목록을 확인하고 선택한 ATM에서의 transaction을 시작할 수 있다.

- /: 모든 Bank, User, Account, ATM, Card 목록을 확인할 수 있다.
- 0: 프로그램이 종료된다.

1-1. Bank

```
================================== Bank ==================================
Please select an option for Bank.
1: Create Bank
2: Use Bank

0: Return to Main Menu
==========================================================================
```

ⓑ bank task를 선택한 경우

Main에서 1번 Bank를 선택하면, ⓑ와 같이 Bank와 관련된 업무를 선택하여 진행할 수 있다. 각 항목에서 진행되는 업무는 다음과 같다.

- 1: 새로운 은행을 생성할 수 있다.
- 2: 생성되어 있는 은행에 대하여 계좌를 개설할 수 있다.

1-1-1. Create Bank

```
================================ Bank ================================
Enter Bank Name (0 to cancel):
```

Bank에서 1을 입력한 경우

ⓑ에서 1: Create Bank를 선택하면, 생성할 Bank의 이름을 입력 받는다.

```
================================ Bank ================================
Enter Bank Name (0 to cancel): Bank
Bank created: Bank
Bank [Bank] created successfully!
======================================================================
```

Create Bank 진행 과정

개설할 Bank의 이름을 입력하니 성공적으로 Bank가 개설되었다. 그리고 다시 ⓑ 화면이 출력된다.

1-1-2. Use Bank

```
================================ Bank ================================
Please select the bank to use.
Available Banks:
1: Bank
2: Bank2

0: Return to Main Menu
======================================================================
```

Bank에서 2를 입력한 경우

ⓑ에서 2: Use Bank를 선택하면, Available Bank의 목록이 출력되면서 사용할 bank를 묻는다.

```
Selected Bank: Bank
============================== Account ===============================
Would you like to create an account with the selected bank?
1: Yes
0: Return to Bank Menu
======================================================================
```

Use Bank 진행 과정 (1)

Bank에서 bank를 선택 시 해당 bank에 대해 Account를 create할 건지 묻는다.

```
============================== Account ===============================
Bank Name: Bank
Enter User Name (0 to cancel):
```

Use Bank 진행 과정 (2)

1: Yes를 선택하면, 개설할 Account의 User name을 입력 받는다.

```
================================== Account ==================================
Bank Name: Bank
Enter User Name (0 to cancel): User
Enter Account Number (12 digits, 0 to cancel):
```

Use Bank 진행 과정 (3)

이어서, 개설할 Account의 Account Number를 입력 받는다.

```
================================ Account ================================
Bank Name: Bank
Enter User Name (0 to cancel): User
Enter Account Number (12 digits, 0 to cancel): 111111111111
Enter Available Funds (0 to cancel):
```

Use Bank 진행 과정 (4)

이어서, 개설할 Account의 Available Fund를 입력 받는다.

```
================================ Account ================================
Bank Name: Bank
Enter User Name (0 to cancel): User
Enter Account Number (12 digits, 0 to cancel): 111111111111
Enter Available Funds (0 to cancel): 5000
Enter Password (4 digits, 0 to cancel):
```

Use Bank 진행 과정 (5)

이어서, 개설할 Account의 Password를 입력 받는다.

```
================================ Account ================================
Bank Name: Bank
Enter User Name (0 to cancel): User
Enter Account Number (12 digits, 0 to cancel): 111111111111
Enter Available Funds (0 to cancel): 5000
Enter Password (4 digits, 0 to cancel): 1111
========================================================================
Card Number:
```

Use Bank 진행 과정 (6)

이어서, 개설할 Account와 연결된 Card Number를 입력 받는다.

```
================================ Account ================================
Bank Name: Bank
Enter User Name (0 to cancel): User
Enter Account Number (12 digits, 0 to cancel): 111111111111
Enter Available Funds (0 to cancel): 5000
Enter Password (4 digits, 0 to cancel): 1111
========================================================================
Card Number: 11111111
========================================================================
Card created successfully for Account: 111111111111
Account created successfully with Bank: Bank
```

Use Bank 진행 과정 (7)

8자리의 Card Number를 입력하니 선택한 bank의 계좌가 개설됨과 동시에 해당 계좌와 연결된 card가 생성되었다.

```
================================ Account ================================
Would you like to create an account with the selected bank?
1: Yes
0: Return to Bank Menu
========================================================================
```

Use Bank 진행 과정 (8)

그리고 선택했던 Bank에 대해 Account를 다시 한 번 추가할 건지 묻는다.

1-2. User

```
================================== User ==================================
Please select an option for User.
1: Create User
2: Use User

0: Return to Main Menu
==========================================================================
```

ⓒ User task를 선택한 경우

Main에서 2번 User를 선택하면, ⓒ와 같이 User와 관련된 업무를 선택하여 진행할 수 있다. 각 항목에서 진행되는 업무는 다음과 같다.

- 1: 새로운 User를 추가할 수 있다.
- 2: 모든 User 목록을 확인할 수 있으며, 선택한 user의 모든 계좌를 열람할 수 있다.

1-2-1. Create User

```
================================== User ==================================
Enter User Name (0 to cancel):
```

User에서 1을 입력한 경우

ⓒ에서 1: Create User를 선택하면, 추가할 User의 이름을 입력 받는다.

```
================================ User ================================
Enter User Name (0 to cancel): User
User [User] created successfully!
======================================================================
```

Create User 진행 과정

User Name을 입력하니 성공적으로 User가 추가되었다. 그리고 다시 ⓒ 화면이 출력된다.

1-2-2. Use User

```
================================ User ================================
Please select the user to use.
Available Users:
1: User

0: Return to Main Menu
======================================================================
```

User에서 2를 입력한 경우

ⓒ에서 2: Use User를 선택하면, Available User 목록이 출력되면서 계좌 목록을 확인할 User를 선택하도록 한다.

```
Selected User: User
User's Accounts:
: Bank Bank, Account No: 111111111111, Balance: 5000
```

Use User 진행 과정

user 목록에서 User를 선택하니 해당 user의 계좌 목록을 보여준다. 그리고 다시 ⓒ 화면이 출력된다.

1-3. ATM

```
================================ ATM =================================
Please select an option for ATM.
1: Create ATM
2: Use ATM

0: Return to Main Menu
======================================================================
```

ⓓ ATM task를 선택한 경우

Main에서 3번 ATM을 선택하면, ⓓ와 같이 ATM과 관련된 업무를 선택하여 진행할 수 있다. 각 항목에서 진행되는 업무는 다음과 같다.

- 1: 새로운 ATM을 생성할 수 있다.
- 2: 생성되어 있는 ATM에 대하여 거래를 진행할 수 있다.

1-3-1. Create ATM

```
Enter Primary Bank Name (0 to cancel):
```

Create ATM 진행 과정 (1)

ATM에서 1번 선택 시 생성할 ATM의 Primary Bank Name을 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel):
```

Create ATM 진행 과정 (2)

이어서, 생성할 ATM의 Serial Number를 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel):
```

Create ATM 진행 과정 (3)

이어서, 생성할 ATM의 Type을 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel):
```

Create ATM 진행 과정 (4)

이어서, 생성할 ATM의 Language Type을 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 2
Enter number of KRW 50000:
```

Create ATM 진행 과정 (5)

이어서, 생성할 ATM의 기초 자금으로서 50000원 권의 개수를 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 2
Enter number of KRW 50000: 5
Enter number of KRW 10000:
```

Create ATM 진행 과정 (6)

이어서, 생성할 ATM의 기초 자금으로서 10000원 권의 개수를 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 2
Enter number of KRW 50000: 5
Enter number of KRW 10000: 5
Enter number of KRW 5000:
```

Create ATM 진행 과정 (7)

이어서, 생성할 ATM의 기초 자금으로서 5000원 권의 개수를 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 2
Enter number of KRW 50000: 5
Enter number of KRW 10000: 5
Enter number of KRW 5000: 5
Enter number of KRW 1000:
```

Create ATM 진행 과정 (8)

이어서, 생성할 ATM의 기초 자금으로서 1000원 권의 개수를 입력 받는다.

```
Enter Primary Bank Name (0 to cancel): Bank
Enter ATM Serial Number (0 to cancel): 111111
Enter ATM Type (1: Single, 2: Multi, 0 to cancel): 1
Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): 2
Enter number of KRW 50000: 5
Enter number of KRW 10000: 5
Enter number of KRW 5000: 5
Enter number of KRW 1000: 5
ATM [Bank: Bank, Serial: 111111] created successfully!
```

Create ATM 진행 과정 (9)

생성할 ATM에 대한 정보를 모두 입력하니 ATM이 성공적으로 생성되었다. 그리고 다시 ⓓ 화면이 출력된다.

1-3-2. Use ATM

```
================================ ATM ================================
Please select the ATM to use.
Available ATMs:
1: ATM [SN: 111111, Bank: Bank, Type: 1]
2: ATM [SN: 222222, Bank: Bank2, Type: 2]

0: Return to Main Menu
=====================================================================
```

Use ATM 진행 과정 (1)

ATM에서 2 선택 시 Available ATM의 목록이 출력되면서 이용할 ATM을 선택하도록 한다.

```
Selected ATM [Bank: Bank, SN: 111111]
=====================================================================
Choose Language
1: English
2: Korean
```

Use ATM 진행 과정 (2)

Bilingual를 지원하는 111111 ATM을 선택하니 언어를 선택하는 화면이 등장한다.

```
1
=====================================================================
Language set to English.
=====================================================================
Please insert your card.
```

Use ATM 진행 과정 (3)

1번 영어를 선택하니 언어가 영어로 설정됨과 동시에 카드를 삽입하라는 메시지가 출력된다.

```
11111111
========================================================================
Enter the card password.
========================================================================
```

Use ATM 진행 과정 (4)

카드 번호를 입력하니 카드의 password를 입력하라는 메시지가 출력된다.

```
========================================================================
1111
========================================================================
Card successfully inserted!
========================================================================
Select a transaction option.
1. Deposit
2. Withdrawal
3. Transfer

0. Return to ATM menu
========================================================================
```

Use ATM 진행 과정 (5)

카드의 password를 바르게 입력하니 카드가 성공적으로 삽입되었다는 메시지가 출력되었다. 동시에 transaction 선택지가 등장한다. 각 선택지에서 진행할 수 있는 업무는 다음과 같다.

- 1: Deposit을 할 수 있다. Deposit은 Cash 또는 Check에 대해 진행 가능하다.
- 2: Withdrawal을 할 수 있다. Withdrawal은 한 session당 최대 3회 가능하다.
- 3: Transfer를 할 수 있다. Transfer는 현금 이체 또는 계좌 이체가 가능하다.
- 0: ⓓ 화면으로 돌아간다.

```
0
========================================================================
The session ended.
========================================================================
No transaction history available for this session.
Returning to main ATM menu.
```

Use ATM 진행 과정 (6)

transaction을 선택하지 않고 0번 ATM menu로 돌아가기를 선택하니 session이 종료됨과 동시에 해당 session의 거래 history를 출력한다. 또한 ⓓ의 화면이 등장한다.

```
Choose Language
1: English
2: Korean
2
========================================================================
언어가 한국어로 설정되었습니다.
========================================================================
카드를 넣어 주세요.
```

Use ATM 진행 과정 (7) - 한국어 선택

언어 선택에서 2번 한국어를 선택하니 메시지가 한국어로 출력된다.

```
11111111
========================================================================
카드 비밀번호를 입력하세요.
========================================================================
1110
========================================================================
틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요.
========================================================================
1110
========================================================================
틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요.
========================================================================
1111
========================================================================
카드가 정상적으로 삽입되었습니다!
========================================================================
거래 옵션을 선택하세요.
1. 입금
2. 출금
3. 이체

0. ATM 메뉴로 돌아가기
========================================================================
```

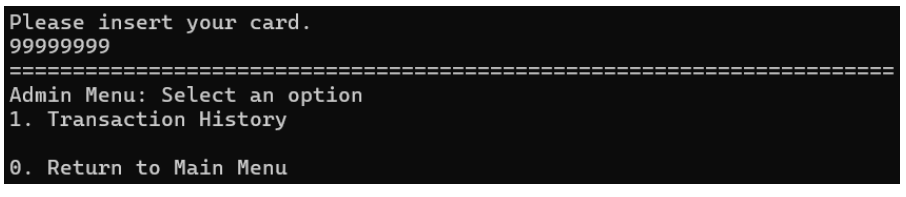
Use ATM 진행 과정 (8)

한국어 선택 후 카드 번호를 입력하고 비밀번호 입력을 3번째 시도에서 성공한 경우에 카드가 성공적으로 삽입되었다는 메시지가 출력되었다. 동시에 거래 유형 선택 화면이 한국어로 등장한다.

```
2
Selected ATM [Bank: Bank2, SN: 222222]
This ATM only suppports English.
========================================================================
Please insert your card.
```

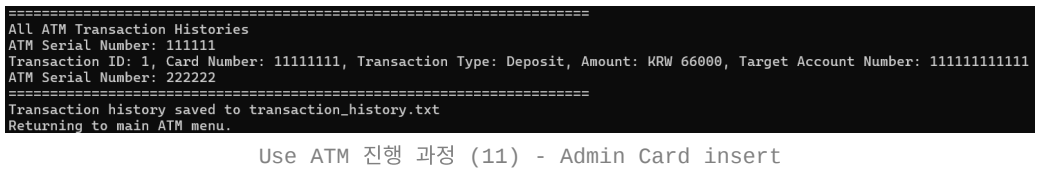
Use ATM 진행 과정 (9)

unilingual를 지원하는 222222 ATM을 선택하니 ‘This ATM only supports English.’라는 메시지가 출력되었다. 그리고 이어서 영어로 된 카드 삽입 요청 메시지가 출력되었다.



Use ATM 진행 과정 (10) - Admin Card insert

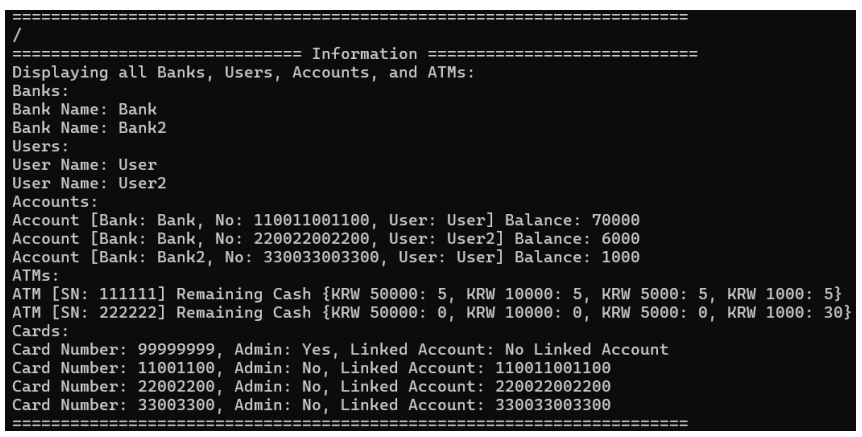
카드 번호 입력 화면에서 admin card로 설정된 카드의 번호인 99999999을 입력하니 Admin Menu 화면이 등장한다.



Use ATM 진행 과정 (11) - Admin Card insert

1번 Transaction History를 선택하니 모든 ATM의 거래 내역이 출력된다. 또한 이 거래 내역은 transaction_history.txt에 저장되어 있다는 메시지가 출력되었다. 그리고 ATM menu로 돌아간다는 메시지와 함께 ⓓ 화면이 출력된다.

1-/. Display all information



Display all information 진행 과정

Main에서 /를 입력한 경우, 현재 등록되어 있는 Bank, User, ATM, Card 목록을 모두 확인 가능하다. 또한 동시에 Main 화면이 등장한다.

7. Final Version of the Source Code

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;

// ============================== Error Type Declaration ==============================

enum ErrorType {
    WrongPassword,
    InvalidCard,
    WrongInput,
    OverDeposit,
    InsufficientFunds,
    ATMInsufficientFunds,
    OverWithdrawPerSession,
    InvalidWithdrawal,
    OverWithdrawal,
    InvalidTransfer,
    InvalidAccount,
    FileError,
    UnusableCheck,
    UnusableCard
};

// ============================ Class Forward Declaration ============================

class User;
class Account;
class Bank;
class Card;
```

```
class AdminCard;
class Transaction;
class Deposit;
class Withdrawal;
class Transfer;
class ATM;

// ====================================== User ======================================

class User {
private:
    string user_name;
    static vector<User*> user_list;
public:
    User(string u_user_name);
    string GetUserName() { return user_name; }
    friend User* FindUser(string u_user_name);
    static vector<User*> GetUserList() { return user_list; }
};

// ====================================== Bank ======================================

class Bank {
private:
    string bank_name;
    static vector<Bank*> bank_list;
public:
    Bank(string b_bank_name);
    string GetBankName() { return bank_name; }
    static vector<Bank*> GetBankList() { return bank_list; }
    friend Bank* FindBank(string b_bank_name);
    Account* ValidateCard(Card* card);
    Account* ValidateAccount(long long int account_number);
};

// ==================================== Account ====================================

class Account {
private:
    long long int account_number;        // 12-digit
    User* user;
    Bank* bank;
    int available_fund;
    int password;        // 4-digit
    static vector<Account*> account_list;
public:
    Account(Bank* bank_ptr, string a_user_name, long long int acc_num, int funds, int pass);
    string GetBankName() { return bank->GetBankName(); }
    long long int GetAccountNumber() { return account_number; }
    int GetPassword() { return password; }
    int GetAvailableFund() { return available_fund; }
    void UpdateAccountMoney(int amount, int mark);
    string GetUserName() { return user ? user->GetUserName() : "Unknown"; }
    static vector<Account*> GetAccountList() { return account_list; }
    static Account* FindAccount(long long int account_number);
};


// ====================================== Card ======================================

class Card {
private:
    int card_number;         // 8-digit
    bool admin = false;
    int card_password;
    Account* linked_account;
protected:
    static vector<Card*> card_list;
public:
    Card(Account* c_linked_account);
    Card();
    void SetCardPassword() { card_password = GetPassword(); }
```

```
    bool GetAdminBool() { return admin; }
    string GetBankName() { return linked_account->GetBankName(); }
    int GetCardNumber() { return card_number; }
    long long int GetAccountNumber() { return linked_account->GetAccountNumber(); }
    int GetPassword() { return linked_account->GetPassword(); }
    void SetAdminBool(bool c_admin) { admin = c_admin; }
    Account* GetAccount() { return linked_account; }
    friend Card* FindCard(int c_card_number);
    void SetCardNumber(int c_card_number) { card_number = c_card_number; }
    static vector<Card*> GetCardList() { return card_list; }
    string ToString() {
        string linked_account_info = linked_account ? to_string(linked_account->GetAccountNumber()) : "No Linked Account
        return "Card Number: " + to_string(card_number) +
            ", Admin: " + ((admin == true) ? "Yes" : "No") +
            ", Linked Account: " + linked_account_info;
    }
};

// ================================== Admin Card ==================================

class AdminCard : public Card {
public:
    AdminCard() : Card() {
        Card::SetCardNumber(99999999);
        SetAdminBool(true);
        card_list.push_back(this);
        cout << "AdminCard created with Card Number: 99999999" << endl;
    }
};

// ================================== Transaction ==================================

class Transaction {
protected:
    string transaction_type;
    ATM* atm;
    int transaction_fee;
    int one_thousand, five_thousand, ten_thousand, fifty_thousand;
    int transaction_amount;
    string additional_info;
    Card* inserted_card;
    Account* source_account;
    Account* target_account;
public:
    Transaction(ATM* atm, Card* inserted_card) : atm(atm), inserted_card(inserted_card) {}
    virtual void SetFee() = 0;
    virtual void GetDeficientFee() {}
    string GetTransactionName() { return transaction_type; }
    int GetCardNumber() { return inserted_card->GetCardNumber(); }
    void SetTransactionAmount() { transaction_amount = one_thousand * 1000 + five_thousand * 5000 + ten_thousand * 10000
    int GetTransactionAmount() { return transaction_amount; }
    Account* GetSourceAccount() { return source_account; }
    Account* GetTargetAccount() { return target_account; }
};

// ================================== Deposit ==================================w

class Deposit : public Transaction {
private:
    int choose_cash;
    int num_of_check;
    long long int sum_of_check = 0;
    long long int incheck;
public:
    Deposit(ATM* d_atm, Card* d_inserted_card) : Transaction(d_atm, d_inserted_card) { transaction_type = "Deposit"; }
    void SetFee();
    void GetDeficientFee();
    void DepositInprogress();
    void GetCash(int one_thousand, int five_thousand, int ten_thousand, int fifty_thousand);
    void GetCheck();
};
```

```
// ================================== Withdrawal ====================================

class Withdrawal : public Transaction {
private:
    const int max_drawal_cash = 500000;
    const int max_drawals_session = 3;
public:
    static int withdrawal_count;
    Withdrawal(ATM* w_atm, Card* w_inserted_card) : Transaction(w_atm, w_inserted_card) { transaction_type = "Withdrawal
    void SetFee();
    void WithdrawalInprogress();
};

// ================================== Transfer ====================================

class Transfer : public Transaction {
private:
    int final_transfer_amount;
    int inserted_amount;
    int final_minus_amount;
    int transfer_type;
    int input_transaction_fee;

public:
    Transfer(ATM* t_atm, Card* t_inserted_card) : Transaction(t_atm, t_inserted_card) { transaction_type = "Transfer"; }
    void GetDeficientFee();
    void SetFee();
    void TransferInprogress();
    void CashTransfer();
    void AccountTransfer();
};

// ====================================== ATM ======================================

class ATM {
private:
    int serial_number;         // 6-digit
    int ATM_type;         // single = 1, multi = 2
    Bank* primary_bank;
    int ATM_lan_type;         // unilingual = 1, bilingual = 2
    static int language_option;         // English = 1, Korean = 2
    int ATM_fund;
    int ATM_1000_num = 0, ATM_5000_num = 0, ATM_10000_num = 0, ATM_50000_num = 0;
    static int session_num;
    static int transaction_num;
    bool card_inserted = false;
    bool admin_access = false;
    vector<vector<string>> ATM_history;
    static vector<ATM*> atm_list;
    Deposit* deposit;
    Transfer* transfer;
    Withdrawal* withdrawal;
public:
    ATM(string bank_name, int serial, int type, int lang, int num_1000, int num_5000, int num_10000, int num_50000);
    void BeforeInterface();
    void CardInsert();
    void AdminInterface();
    void SessionStarts();
    void SessionEnds();
    void UpdateATMFund(int a_one_thousand, int a_five_thousand, int a_ten_thousand, int a_fifty_thousand, int mark);
    void AddATMHistory(Transaction* transaction);
    void ShowSessionHistory();
    void ShowTransactionHistory();
    void SetCardInserted(bool inserted) { card_inserted = inserted; }
    void SetAdminAccess(bool access) { admin_access = access; }
    void SetATMFund(int a_one_thousand, int a_five_thousand, int a_ten_thousand, int a_fifty_thousand) { ATM_fund = a_on
    int GetATMFund() { return ATM_fund; }
    int GetFiftyThousand() { return ATM_50000_num; }
    int GetTenThousand() { return ATM_10000_num; }
    int GetFiveThousand() { return ATM_5000_num; }
    int GetOneThousand() { return ATM_1000_num; }
    int GetSessionNum() { return session_num; }
```

```
    string GetPrimaryBankName() { return primary_bank->GetBankName(); }
    static int GetLanguageOption() { return language_option; }
    void StartInterface(Card* card);
    void TransactionHistorytoFile();
    void ShowError(ErrorType errortype);
    void ShowSnapshot();
    static vector<ATM*> GetATMList() { return atm_list; }
    int GetSerialNumber() { return serial_number; }
    int GetATMType() { return ATM_type; }
};

// ================================ List Initialization ================================

vector<ATM*> ATM::atm_list = {};
vector<Bank*> Bank::bank_list = {};
vector<Account*> Account::account_list = {};
vector<User*> User::user_list = {};
vector<Card*> Card::card_list = {};

// ============================ User External Declaration ============================

User::User(string u_user_name) {
    user_name = u_user_name;
    user_list.push_back(this);
}

User* FindUser(string u_user_name) {
    for (size_t i = 0; i < User::user_list.size(); i++) {
        if (u_user_name == User::user_list[i]->GetUserName()) {
            return User::user_list[i];
        }
    }
    return nullptr;
}

// ============================ Bank External Declaration ============================

Bank::Bank(string b_bank_name) {
    bank_name = b_bank_name;
    bank_list.push_back(this);
}

Bank* FindBank(string b_bank_name) {
    for (size_t i = 0; i < Bank::bank_list.size(); i++) {
        if (b_bank_name == Bank::bank_list[i]->GetBankName()) {
            return Bank::bank_list[i];
        }
    }
    return nullptr;
}

// ========================== Account External Declaration ==========================

Account::Account(Bank* bank_ptr, string a_user_name, long long int acc_num, int funds, int pass)
    : bank(bank_ptr), account_number(acc_num), available_fund(funds), password(pass) {
    user = FindUser(a_user_name);
    if (!user) {
        cout << "User does not exist. Please create the user first." << endl;
        return;
    }

    try {
        Card* new_card = new Card(this);
        if (new_card) {
            cout << "Card created successfully for Account: " << this->GetAccountNumber() << endl;
        }
    }
    catch (const invalid_argument& e) {
        cout << "Account creation aborted due to invalid card number. Please try again." << endl;
        throw;
    }
    account_list.push_back(this);
```

```
}

Account* Account::FindAccount(long long int account_number) {
    for (auto& account : Account::account_list) {
        if (account->GetAccountNumber() == account_number) {
            return account;
        }
    }
    return nullptr;
}

Account* Bank::ValidateAccount(long long int account_number) {
    return Account::FindAccount(account_number);
}

Account* Bank::ValidateCard(Card* card) {
    Account* valid_account = card->GetAccount();
    if (FindBank(valid_account->GetBankName()) == nullptr) {
        return nullptr;
    }
    if (Account::FindAccount(valid_account->GetAccountNumber()) == nullptr) {
        return nullptr;
    }
    int inpassword;
    int try_num = 0;
    do {
        if (try_num == 0) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Enter the card password." << endl;
                cout << "================================================================================" << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "카드 비밀번호를 입력하세요." << endl;
                cout << "================================================================================" << endl;
            }
        }
        else {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Wrong card password. Please enter the card password again." << endl;
                cout << "================================================================================" << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "틀린 비밀번호입니다. 카드 비밀번호를 다시 입력하세요." << endl;
                cout << "================================================================================" << endl;
            }
        }
        cin >> inpassword;
        if (inpassword == card->GetPassword()) {
            return valid_account;
        }
        try_num++;
    } while (try_num < 3);
    return nullptr;
}

void Account::UpdateAccountMoney(int amount, int mark) {
    if (mark == 1) {
        this->available_fund += amount;
    }
    else {
        this->available_fund -= amount;
    }
}

// ============================ Card External Declaration ============================

Card::Card() {
```

```
    card_number = 0;
    card_password = 0;
    linked_account = nullptr;
}

Card::Card(Account* c_linked_account) {
    if (!admin) {
        cout << "================================================================================" << endl;
        cout << "Card Number: ";
        int c_card_number;
        cin >> c_card_number;
        if (to_string(c_card_number).length() != 8) {
            cout << "Invalid Card Number. Please enter an 8-digit number." << endl;
            cout << "================================================================================" << endl;
            throw invalid_argument("Invalid Card Number");
        }
        card_number = c_card_number;

        linked_account = c_linked_account;
        if (!linked_account) {
            cout << "Error: Linked account is null!" << endl;
            return;
        }

        SetCardPassword();
        cout << "================================================================================" << endl;
        card_list.push_back(this);
    }
}

Card* FindCard(int c_card_number) {
    for (size_t i = 0; i < Card::card_list.size(); i++) {
        if (c_card_number == Card::card_list[i]->GetCardNumber()) {
            return Card::card_list[i];
        }
    }
    return nullptr;
}

// ============================ Deposit External Declaration ============================

void Deposit::SetFee() {
    if (target_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else { transaction_fee = 2000; }
}

void Deposit::GetDeficientFee() {
    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Please put in the appropriate fee as bills." << endl;
        cout << "(Deposit fee for primary banks: KRW 1000, Deposit fee for non-primary banks: KRW 2000)" << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "수수료를 현금으로 납부해 주세요." << endl;
        cout << "(주거래 은행 예치 요금: 1000원, 주거래 은행 외 예치 요금: 2000원)" << endl;
        cout << "================================================================================" << endl;
    }
    int inserted_fee;
    cin >> inserted_fee;
    int d_one_thousand = transaction_fee / 1000;
    if (transaction_fee > inserted_fee) {
        while (transaction_fee > inserted_fee) {
            transaction_fee = transaction_fee - inserted_fee;
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Please enter proper fee. (left: KRW " << transaction_fee << ")" << endl;
                cout << "================================================================================" << endl;
            }
```

```
            else {
                cout << "================================================================================" << endl;
                cout << "정확한 수수료 금액을 넣어 주세요. (남은 금액: " << transaction_fee << ")" << endl;
                cout << "================================================================================" << endl;
            }
            cin >> inserted_fee;
        }
    }
    atm->UpdateATMFund(d_one_thousand, 0, 0, 0, 1);
}

void Deposit::DepositInprogress() {
    target_account = inserted_card->GetAccount();
    if (to_string(target_account->GetAccountNumber()).length() != 12) {
        atm->ShowError(InvalidAccount);
    }
    SetFee();
    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Which one do you want to deposit, cash or check?" << endl;
        cout << "1. Cash" << endl;
        cout << "2. Check\n" << endl;
        cout << "0. Cancel" << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "입금 유형을 선택하세요." << endl;
        cout << "1. 현금" << endl;
        cout << "2. 수표\n" << endl;
        cout << "0. 취소" << endl;
        cout << "================================================================================" << endl;
    }
    cin >> choose_cash;
    if (choose_cash == 1) {
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Please put in as many bills as you want.\n" << endl;
            cout << "Enter number of KRW 50000: "; cin >> fifty_thousand;
            cout << "Enter number of KRW 10000: "; cin >> ten_thousand;
            cout << "Enter number of KRW 5000: "; cin >> five_thousand;
            cout << "Enter number of KRW 1000: "; cin >> one_thousand;
            cout << "================================================================================" << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "입금하실 현금을 넣어 주세요.\n" << endl;
            cout << "50000원권 지폐 수를 입력하세요: "; cin >> fifty_thousand;
            cout << "10000원권 지폐 수를 입력하세요: "; cin >> ten_thousand;
            cout << "5000원권 지폐 수를 입력하세요: "; cin >> five_thousand;
            cout << "1000원권 지폐 수를 입력하세요: "; cin >> one_thousand;
        }
        GetCash(one_thousand, five_thousand, ten_thousand, fifty_thousand);
    }
    else if (choose_cash == 2) {
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Please enter the check up to 30 papers." << endl;
            cout << "If you want to stop entering the check, press 0." << endl;
            cout << "================================================================================" << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "수표는 최대 30장까지 넣을 수 있습니다." << endl;
            cout << "수표 입력을 중지하려면 '0'을 눌러 주세요." << endl;
            cout << "================================================================================" << endl;
        }
        GetCheck();
    }
    else if (choose_cash == 0) { atm->SessionEnds(); }
    else {
        atm->ShowError(WrongInput);
```

```
        return;
    }
    atm->AddATMHistory(this);
}

void Deposit::GetCash(int d_one_thousand, int d_five_thousand, int d_ten_thousand, int d_fifty_thousand) {
    int num_of_cash = d_one_thousand + d_five_thousand + d_ten_thousand + d_fifty_thousand;
    SetTransactionAmount();

    if (num_of_cash > 50) {
        atm->ShowError(OverDeposit);
    }
    else {
        GetDeficientFee();
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Deposit proceeded successfully." << endl;
            cout << "================================================================================" << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "입금이 정상적으로 진행되었습니다." << endl;
            cout << "================================================================================" << endl;
        }
        atm->UpdateATMFund(d_one_thousand, d_five_thousand, d_ten_thousand, d_fifty_thousand, 1);
        target_account->UpdateAccountMoney(transaction_amount, 1);
    }
}

void Deposit::GetCheck() {
    for (num_of_check = 0; num_of_check < 31; num_of_check++) {
        cin >> incheck;
        if (incheck == 0) {
            break;
        }
        else if (num_of_check >= 30) {
            atm->ShowError(OverDeposit);
            return;
        }
        else if (incheck < 100000) {
            atm->ShowError(UnusableCheck);
            return;
        }
        sum_of_check = sum_of_check + incheck;
    }

    transaction_amount = sum_of_check;
    GetDeficientFee();

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Deposit proceeded successfully." << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "입금이 정상적으로 진행되었습니다." << endl;
        cout << "================================================================================" << endl;
    }
    target_account->UpdateAccountMoney(transaction_amount, 1);
    return;
}

// ========================= Witdrawal External Declaration ==========================

int Withdrawal::withdrawal_count = 0;

void Withdrawal::SetFee() {
    if (source_account->GetBankName() == atm->GetPrimaryBankName()) {
        transaction_fee = 1000;
    }
    else {
```

```
        transaction_fee = 2000;
    }
}

void Withdrawal::WithdrawalInprogress() {
    while (true) {
        if (withdrawal_count == 3) {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Session limit reached.\nPlease start a new session for further withdrawals." << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "세션당 출금은 최대 3회만 가능합니다.\n추가 출금을 원하신다면, 새로운 세션을 시작해주세요." << endl;
            }
            atm->SessionEnds();
            break;
        }
        else {
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "The maximum number of withdrawals allowed per session is 3, with a transaction limit of KRW 500
                cout << withdrawal_count << " withdrawals have been made in this session." << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "한 세션당 최대 3회의 출금이 가능하며, 거래 한도는 500000원입니다.\n수표로는 출금이 불가능하니 유의해주십시오." <<
                cout << "현재 해당 세션에서 출금이 " << withdrawal_count << "번 진행됐습니다." << endl;
            }
        }
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Please enter the withdrawal amount." << endl;
            cout << "If you would like to cancel the transaction, please press 0." << endl;
            cout << "================================================================================" << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "출금하실 금액을 입력해주세요." << endl;
            cout << "거래 취소를 원하신다면, 0을 입력해주세요." << endl;
            cout << "================================================================================" << endl;
        }
        cin >> transaction_amount;
        if (transaction_amount == 0) {
            return;
        }

        source_account = inserted_card->GetAccount();
        if (source_account == nullptr) {
            atm->ShowError(InvalidAccount);
        }

        if (transaction_amount < 0) {
            atm->ShowError(InvalidWithdrawal);
            return;
        }
        if (transaction_amount > max_drawal_cash) {
            atm->ShowError(OverWithdrawal);
            return;
        }
        if (source_account->GetAvailableFund() < (transaction_amount + transaction_fee)) {
            atm->ShowError(InsufficientFunds);
            return;
        }

        SetFee();
        int ATM_fifty_thousand = atm->GetFiftyThousand();
        int ATM_ten_thousand = atm->GetTenThousand();
        int ATM_five_thousand = atm->GetFiveThousand();
        int ATM_one_thousand = atm->GetOneThousand();

        int confirm_amount = transaction_amount;
```

```
int remaining_amount = transaction_amount;
fifty_thousand = min(remaining_amount / 50000, ATM_fifty_thousand);
remaining_amount -= fifty_thousand * 50000;
ten_thousand = min(remaining_amount / 10000, ATM_ten_thousand);
remaining_amount -= ten_thousand * 10000;
five_thousand = min(remaining_amount / 5000, ATM_five_thousand);
remaining_amount -= five_thousand * 5000;
one_thousand = min(remaining_amount / 1000, ATM_one_thousand);
remaining_amount -= one_thousand * 1000;

if (!(confirm_amount % 1000 == 0)) {
    atm->ShowError(WrongInput);
    return;
}

if (remaining_amount > 0) {
    atm->ShowError(ATMInsufficientFunds);
    return;
}

if (ATM::GetLanguageOption() == 1) {
    cout << "================================================================================" << endl;
            cout << "You have successfully withdrawn: KRW " << transaction_amount << endl;
            cout << "Withdrawal fee: KRW " << transaction_fee << endl;
            cout << "Fees have been paid from the withdrawal account." << endl;
        }
        else {
            cout << "================================================================================" << endl;
            cout << "성공적으로 출금되었습니다: " << transaction_amount << "원" << endl;
            cout << "출금 수수료: " << transaction_fee << "원" << endl;
            cout << "수수료가 출금 계좌에서 차감되었습니다." << endl;
        }

        source_account->UpdateAccountMoney(transaction_amount + transaction_fee, 0);
        atm->UpdateATMFund(one_thousand, five_thousand, ten_thousand, fifty_thousand, 0);
        withdrawal_count++;
        atm->AddATMHistory(this);
        return;
    }
}

// ========================== Transfer External Declaration ==========================

void Transfer::SetFee() {
    if (transfer_type == 2) {
        if (source_account->GetBankName() == atm->GetPrimaryBankName() && target_account->GetBankName() == atm->GetPrima
            transaction_fee = 2000;
        }
        else if (source_account->GetBankName() != atm->GetPrimaryBankName() && target_account->GetBankName() != atm->Get
            transaction_fee = 4000;
        }
        else {
            transaction_fee = 3000;
        }
    }
    else if (transfer_type == 1) {
        transaction_fee = 1000;
    }
}

void Transfer::GetDeficientFee() {
    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Please put in the appropriate fee as bills." << endl;
        cout << "(Cash transfer fee between primary banks: KRW 2000, Cash transfer fee between primary and non-primary b
        cout << "Cash transfer fee between non-primary banks: KRW 4000)" << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "수수료를 현금으로 납부해 주세요." << endl;
```

```
        cout << "(주거래 은행 계좌 간 이체 요금: KRW 2000, 주거래 은행 계좌와 주거래 은행 외 계좌 간 이체 요금: KRW 3000," << endl;
        cout << "주거래 은행 외 계좌 간 이체 요금: KRW 4000)" << endl;
        cout << "================================================================================" << endl;
    }
    int inserted_fee;
    cin >> inserted_fee;
    int d_one_thousand = transaction_fee / 1000;
    if (transaction_fee > inserted_fee) {
        while (transaction_fee > inserted_fee) {
            transaction_fee = transaction_fee - inserted_fee;
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Please enter proper fee. (left: KRW " << transaction_fee << ")" << endl;
                cout << "================================================================================" << endl;
            }
            else {
                cout << "================================================================================" << endl;
                cout << "정확한 수수료 금액을 넣어 주세요. (남은 금액: " << transaction_fee << ")" << endl;
                cout << "================================================================================" << endl;
            }
            cin >> inserted_fee;
        }
    }
    atm->UpdateATMFund(d_one_thousand, 0, 0, 0, 1);
}

void Transfer::TransferInprogress() {
    if (!inserted_card) {
        atm->ShowError(InvalidCard);
        return;
    }
    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Choose transfer type. \n1. Cash Transfer\n2. Account Transfer\n\n0. Transaction cancellation\n";
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "이체 유형을 선택하세요. \n1. 현금 이체\n2. 계좌 이체\n\n0. 거래 취소\n";
        cout << "================================================================================" << endl;
    }
    cin >> transfer_type;

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Enter the target account number.\n";
        cout << "If you would like to cancel the transaction, please press 0." << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "받는 분의 계좌번호를 입력하세요.\n";
        cout << "거래 취소를 원하신다면, 0을 입력해주세요." << endl;
        cout << "================================================================================" << endl;
    }
    long long int input_target_account_number;
    if (!(cin >> input_target_account_number)) {
        atm->ShowError(WrongInput);
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return;
    }

    if (input_target_account_number == 0) {
        atm->SessionEnds();
    }

    target_account = Account::FindAccount(input_target_account_number);
    if (!target_account) {
        atm->ShowError(InvalidAccount);
        return;
    }
```

```
    if (transfer_type == 1) {
        transaction_type = "Cash Transfer";
        CashTransfer();
    }
    else if (transfer_type == 2) {
        transaction_type = "Account Transfer";
        AccountTransfer();
    }
    else if (transfer_type == 0) {
        atm->SessionEnds();
    }
    else {
        atm->ShowError(WrongInput);
    }
    atm->AddATMHistory(this);
}

void Transfer::CashTransfer() {
    SetFee();
    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Please insert cash for each bill denomination in accordance with the transfer amount.\n";
        cout << "Enter number of KRW 50000: "; cin >> fifty_thousand;
        cout << "Enter number of KRW 10000: "; cin >> ten_thousand;
        cout << "Enter number of KRW 5000: "; cin >> five_thousand;
        cout << "Enter number of KRW 1000: "; cin >> one_thousand;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "이체할 금액에 맞게 각 지폐 단위별로 현금을 입금해주세요.\n";
        cout << "50000원권 지폐 수를 입력하세요: "; cin >> fifty_thousand;
        cout << "10000원권 지폐 수를 입력하세요: "; cin >> ten_thousand;
        cout << "5000원권 지폐 수를 입력하세요: "; cin >> five_thousand;
        cout << "1000원권 지폐 수를 입력하세요: "; cin >> one_thousand;
    }
    SetTransactionAmount();
    GetDeficientFee();

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "KRW " << transaction_amount << " will be transferred." << endl;
        cout << "1. Transfer amount is correct.\n\n0: End Session" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "수수료를 제외한 " << transaction_amount << "원이 이체됩니다." << endl;
        cout << "1. 이체 금액이 맞습니다.\n\n0: 거래 종료" << endl;
    }
    int transfer_confirm;
    cin >> transfer_confirm;
    if (transfer_confirm == 1) {
        if (transaction_amount > 0) {
            atm->UpdateATMFund(one_thousand + input_transaction_fee, five_thousand, ten_thousand, fifty_thousand, 1);
            target_account->UpdateAccountMoney(transaction_amount, 1);
            if (ATM::GetLanguageOption() == 1) {
                cout << "================================================================================" << endl;
                cout << "Cash transfer successful. KRW " << transaction_amount << " transferred.\n";
            }
            else {
                cout << "================================================================================" << endl;
                cout << "현금 이체가 완료되었습니다. " << transaction_amount << "원이 이체되었습니다.\n";
            }
        }
        else {
            atm->ShowError(InvalidTransfer);
        }
    }
    else if (transfer_confirm == 0) {
        atm->SessionEnds();
    }
    else {
```

```
        atm->ShowError(WrongInput);
    }
}

void Transfer::AccountTransfer() {
    source_account = inserted_card->GetAccount();
    SetFee();

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "Enter amount to transfer.\nIf you would like to cancel the transaction, please press 0." << endl;
        cout << "================================================================================" << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "이체할 금액을 입력하세요.\n거래 취소를 원하신다면, 0을 입력해주세요.";
        cout << "================================================================================" << endl;
    }
    cin >> transaction_amount;
    if (transaction_amount == 0) {
        atm->SessionEnds();
    }
    final_minus_amount = transaction_amount + transaction_fee;

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "The transaction fee is KRW " << transaction_fee << "." << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << "거래 수수료는 " << transaction_fee << "입니다." << endl;
    }

    if (ATM::GetLanguageOption() == 1) {
        cout << "================================================================================" << endl;
        cout << "KRW " << final_minus_amount
            << " will be deducted from account " << source_account->GetAccountNumber()
            << ", and KRW " << transaction_amount << " will be deposited into account "
            << target_account->GetAccountNumber() << "." << endl;
    }
    else {
        cout << "================================================================================" << endl;
        cout << source_account->GetAccountNumber() << " 계좌에서 " << final_minus_amount
            << "원이 차감되며, " << target_account->GetAccountNumber() << " 계좌로 "
            << transaction_amount << "원이 입금됩니다." << endl;
    }

    if (source_account->GetAvailableFund() >= final_minus_amount) {
        source_account->UpdateAccountMoney(final_minus_amount, 0);
        target_account->UpdateAccountMoney(transaction_amount, 1);
        if (ATM::GetLanguageOption() == 1) {
            cout << "================================================================================" << endl;
            cout << "Account transfer successful. " << transaction_amount << " transferred.\n";
        }
        else {
            cout << "================================================================================" << endl;
            cout << "계좌 이체가 완료되었습니다. " << transaction_amount << "원이 이체되었습니다.\n";
        }
    }
    else {
        atm->ShowError(InsufficientFunds);
    }
}

// ============================== ATM External Declaration ==============================

int ATM::language_option = 1;
int ATM::session_num = 0;
int ATM::transaction_num = 1;

ATM::ATM(string bank_name, int serial, int type, int lang, int num_1000, int num_5000, int num_10000, int num_50000) {
    primary_bank = FindBank(bank_name);
```

```
    if (type == 1 && !primary_bank) {
        cout << "Invalid bank name! The card in primary bank is only usable." << endl;
        return;
    }
    serial_number = serial;
    ATM_type = type;
    ATM_lan_type = lang;
    ATM_1000_num = num_1000;
    ATM_5000_num = num_5000;
    ATM_10000_num = num_10000;
    ATM_50000_num = num_50000;
    SetATMFund(num_1000, num_5000, num_10000, num_50000);
    atm_list.push_back(this);
}

void ATM::BeforeInterface() {
    if (ATM_lan_type == 2) {
        cout << "================================================================================" << endl;
        cout << "Choose Language\n1: English\n2: Korean \n";

        cin >> language_option;

        if (language_option == 1) {
            cout << "================================================================================" << endl;
            cout << "Language set to English." << endl;
        }
        else if (language_option == 2) {
            cout << "================================================================================" << endl;
            cout << "언어가 한국어로 설정되었습니다." << endl;
        }
        else {
            ShowError(WrongInput);
            return;
        }
        CardInsert();

        cout << "Returning to main ATM menu." << endl;
        return;
    }
    else {
        language_option = 1;
        cout << "This ATM only supports English." << endl;
        CardInsert();

        cout << "Returning to main ATM menu." << endl;
        return;
    }
}

void ATM::CardInsert() {
    if (language_option == 1) {
        cout << "================================================================================" << endl;
        cout << "Please insert your card." << endl;
    }
    else if (language_option == 2) {
        cout << "================================================================================" << endl;
        cout << "카드를 넣어 주세요." << endl;
    }
    int inserted_card_number;
    cin >> inserted_card_number;
    Card* card = FindCard(inserted_card_number);

    if (card == nullptr) {
        ShowError(InvalidCard);
        if (language_option == 1) {
            cout << "Your card is returned." << endl;
        }
        else if (language_option == 2) {
            cout << "카드가 반환되었습니다." << endl;
        }
        return;
    }
```

```
    if (card->GetAdminBool() == true) {
        SetAdminAccess(true);
        AdminInterface();
        return;
    }
    Bank* valid_bank = FindBank(card->GetBankName());
    if (ATM_type == 1) {
        if (primary_bank->GetBankName() != card->GetBankName()) {
            ShowError(UnusableCard);
            if (language_option == 1) {
                cout << "Your card is returned." << endl;
            }
            else if (language_option == 2) {
                cout << "카드가 반환되었습니다." << endl;
            }
            return;
        }
    }
    Account* valid_account = valid_bank->ValidateCard(card);
    if (valid_account == nullptr) {
        ShowError(WrongPassword);
        if (language_option == 1) {
            cout << "Your card is returned." << endl;
        }
        else if (language_option == 2) {
            cout << "카드가 반환되었습니다." << endl;
        }
        return;
    }
    SetCardInserted(true);
    if (language_option == 1) {
        cout << "================================================================================" << endl;
        cout << "Card successfully inserted!" << endl;
    }
    else if (language_option == 2) {
        cout << "================================================================================" << endl;
        cout << "카드가 정상적으로 삽입되었습니다!" << endl;
    }
    SessionStarts();
    StartInterface(card);
}

void ATM::AdminInterface() {
    if (language_option == 1) {
        cout << "================================================================================" << endl;
        cout << "Admin Menu: Select an option\n1. Transaction History\n\n0. Return to Main Menu" << endl;
    }
    else if (language_option == 2) {
        cout << "================================================================================" << endl;
        cout << "관리자 메뉴: 옵션을 선택하세요.\n1. 거래 내역 보기\n\n0. 메인으로 돌아가기" << endl;
    }
    int choice;
    cin >> choice;
    if (choice == 1) { ShowTransactionHistory(); TransactionHistorytoFile(); }
    else if (choice == 0) { return; }
    else { ShowError(WrongInput); }
    return;
}

void ATM::SessionStarts() {
    session_num++;
}

void ATM::SessionEnds() {
    SetAdminAccess(false);
    SetCardInserted(false);
    if (language_option == 1) {
        cout << "================================================================================" << endl;
        cout << "The session ended." << endl;
    }
    else if (language_option == 2) {
        cout << "================================================================================" << endl;
```

```
        cout << "세션이 종료되었습니다." << endl;
    }
    cout << "================================================================================" << endl;
    ShowSessionHistory();
    Withdrawal::withdrawal_count = 0;
}

void ATM::UpdateATMFund(int a_one_thousand, int a_five_thousand, int a_ten_thousand, int a_fifty_thousand, int mark) {
    if (mark == 1) {
        ATM_1000_num += a_one_thousand;
        ATM_5000_num += a_five_thousand;
        ATM_10000_num += a_ten_thousand;
        ATM_50000_num += a_fifty_thousand;
    }
    else {
        ATM_1000_num -= a_one_thousand;
        ATM_5000_num -= a_five_thousand;
        ATM_10000_num -= a_ten_thousand;
        ATM_50000_num -= a_fifty_thousand;
    }
    SetATMFund(ATM_1000_num, ATM_5000_num, ATM_10000_num, ATM_50000_num);
    return;
}

void ATM::AddATMHistory(Transaction* transaction) {
    if (ATM_history.size() <= session_num) {
        ATM_history.resize(session_num + 1);
    }

    string history = ((language_option == 1) ? "Transaction ID: " : "거래 번호: ") + to_string(transaction_num) +
        ((language_option == 1) ? ", Card Number: " : ", 카드 번호: ") + to_string(transaction->GetCardNumber()) +
        ((language_option == 1) ? ", Transaction Type: " : ", 거래 유형: ") + transaction->GetTransactionName() +
        ((language_option == 1) ? ", Amount: KRW " : ", 금액: ") + to_string(transaction->GetTransactionAmount());

    if (transaction->GetTransactionName() == "Deposit") {
        history += ", Target Account Number: " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Withdrawal") {
        history += ", Source Account Number: " + to_string(transaction->GetSourceAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Cash Transfer") {
        history += ", Target Account Number : " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }
    else if (transaction->GetTransactionName() == "Account Transfer") {
        history += ", Source Account Number: " + to_string(transaction->GetSourceAccount()->GetAccountNumber()) +
            ", Target Account Number: " + to_string(transaction->GetTargetAccount()->GetAccountNumber());
    }

    ATM_history[session_num].push_back(history);
    transaction_num++;
}

void ATM::ShowSessionHistory() {
    if (session_num >= ATM_history.size() || ATM_history[session_num].empty()) {
        if (language_option == 1) {
            cout << "No transaction history available for this session." << endl;
        }
        else if (language_option == 2) {
            cout << "해당 세션에서의 거래 내역이 없습니다." << endl;
        }
        return;
    }

    cout << "================================================================================" << endl;
    cout << ((language_option == 1) ? "Session Transaction History" : "세션 거래 내역") << endl;
    for (const string& record : ATM_history[session_num]) {
        cout << record << endl;
    }
    cout << "================================================================================" << endl;
}

void ATM::ShowTransactionHistory() {
```

```
    cout << "============================================================================" << endl;
    cout << ((language_option == 1) ? "All ATM Transaction Histories" : "모든 ATM 거래 내역") << endl;

    for (auto atm : ATM::GetATMList()) {
        cout << ((language_option == 1) ? "[ATM SN: " : "[ATM 일련번호: ") << atm->serial_number << "]" << endl;
        if (atm->ATM_history.size() == 0) {
            cout << "\tNo transactions in this ATM" << endl;
        }
        for (size_t i = 0; i < atm->ATM_history.size(); i++) {
            for (size_t j = 0; j < atm->ATM_history[i].size(); j++) {
                cout << "\t" << atm->ATM_history[i][j] << endl;
            }
        }
    }
    cout << "============================================================================" << endl;
}

void ATM::StartInterface(Card* card) {
    while (card_inserted == true) {        // if the user does not explicitly end the session
        if (language_option == 1) {
            cout << "============================================================================" << endl;
            cout << "Select a transaction option." << endl;
            cout << "1. Deposit" << endl;
            cout << "2. Withdrawal" << endl;
            cout << "3. Transfer\n" << endl;
            cout << "0. Return to ATM menu" << endl;
            cout << "============================================================================" << endl;
        }
        else if (language_option == 2) {
            cout << "============================================================================" << endl;
            cout << "거래 옵션을 선택하세요." << endl;
            cout << "1. 입금" << endl;
            cout << "2. 출금" << endl;
            cout << "3. 이체\n" << endl;
            cout << "0. ATM 메뉴로 돌아가기" << endl;
            cout << "============================================================================" << endl;
        }

        string transaction_option;
        cin >> transaction_option;


        if (transaction_option == "1") {
            deposit = new Deposit(this, card);
            deposit->DepositInprogress();
            delete deposit;
        }
        else if (transaction_option == "2") {
            withdrawal = new Withdrawal(this, card);
            withdrawal->WithdrawalInprogress();
            delete withdrawal;
        }
        else if (transaction_option == "3") {
            transfer = new Transfer(this, card);
            transfer->TransferInprogress();
            delete transfer;
        }
        else if (transaction_option == "0") {
            SessionEnds();
            return;
        }
        else if (transaction_option == "/") {
            ShowSnapshot();
        }
        else {
            ShowError(WrongInput);
        }
    }
    SetCardInserted(true);
}

void ATM::TransactionHistorytoFile() {
```

```
    ofstream file("transaction_history.txt");
    if (file.is_open()) {
        file << "All ATM Transaction Histories" << endl;
        for (auto atm : ATM::GetATMList()) {
            file << ((language_option == 1) ? "[ATM SN: " : "[ATM 일련번호: ") << atm->serial_number << "]" << endl;
            if (atm->ATM_history.size() == 0) {
                file << "\tNo transactions in this ATM" << endl;
            }
            for (size_t i = 0; i < atm->ATM_history.size(); i++) {
                for (size_t j = 0; j < atm->ATM_history[i].size(); j++) {
                    file << "\t" << atm->ATM_history[i][j] << endl;
                }
            }
        }
        file.close();
        if (language_option == 1) { cout << "Transaction history saved to transaction_history.txt" << endl; }
        else if (language_option == 2) { cout << "거래 내역이 transaction_history.txt에 저장되었습니다." << endl; }
    }
    else {
        ShowError(FileError);
        return;
    }
}

void ATM::ShowError(ErrorType ShowErrortype) {
    cout << "================================error================================" << endl;
    if (language_option == 1) {
        switch (ShowErrortype) {
        case WrongPassword: cout << "Error: Incorrect password." << endl; break;
        case InvalidCard: cout << "Error: Invalid card." << endl; break;
        case InvalidAccount: cout << "Error: Invalid account number." << endl; break;
        case WrongInput: cout << "Error: Invalid input." << endl; break;
        case OverDeposit: cout << "Error: Deposit limit exceeded." << endl; break;
        case InsufficientFunds: cout << "Error: Insufficient account balance." << endl; break;
        case ATMInsufficientFunds: cout << "Error: Insufficient ATM balance." << endl; break;
        case OverWithdrawPerSession: cout << "Error: Withdrawal limit for this session reached. Please start a new sessi
        case InvalidWithdrawal: cout << "Error: Invalid withdrawal amount." << endl; break;
        case OverWithdrawal: cout << "Error: Single deposit limit of 500000 won exceeded." << endl; break;
        case InvalidTransfer: cout << "Error: Invalid transfer amount." << endl; break;
        case FileError: cout << "Error: Unable to open file." << endl; break;
        case UnusableCheck: cout << "Error: Unusable Check. Only over 100000 won of check can be used." << endl; break;
        case UnusableCard: cout << "Error: Unusable Card for this ATM." << endl; break;
        default: cout << "Error: Unknown Error" << endl;
        }
    }
    else if (language_option == 2) {
        switch (ShowErrortype) {
        case WrongPassword: cout << "오류: 잘못된 비밀번호입니다." << endl; break;
        case InvalidCard: cout << "오류: 유효하지 않은 카드입니다." << endl; break;
        case InvalidAccount: cout << "오류: 유효하지 않은 계좌번호입니다." << endl; break;
        case WrongInput: cout << "오류: 잘못된 입력입니다." << endl; break;
        case OverDeposit: cout << "오류: 예치 한도가 초과되었습니다." << endl; break;
        case InsufficientFunds: cout << "오류: 계좌 잔액이 부족합니다." << endl; break;
        case ATMInsufficientFunds: cout << "오류: ATM 잔액이 부족합니다." << endl; break;
        case OverWithdrawPerSession: cout << "오류: 세션 당 출금 한도에 도달하였습니다. 추가 출금을 원하신다면 새로운 세션을 시작해주세요
        case InvalidWithdrawal: cout << "오류: 올바르지 않은 출금 금액입니다. " << endl; break;
        case OverWithdrawal: cout << "오류: 1회 예치 한도 500000원을 초과하였습니다. " << endl; break;
        case InvalidTransfer: cout << "오류: 올바르지 않은 이체 금액입니다." << endl; break;
        case FileError: cout << "오류: 파일을 열 수 없습니다." << endl; break;
        case UnusableCheck: cout << "오류: 사용할 수 없는 수표입니다. 100000원 이상의 수표만 사용 가능합니다." << endl; break;
        case UnusableCard: cout << "오류: 해당 ATM에서 사용할 수 없는 카드입니다." << endl; break;
        default: cout << "오류: 알 수 없는 오류입니다." << endl;
        }
    }
    if (card_inserted == true) {
        SessionEnds();
    }
}

void ATM::ShowSnapshot() {
    cout << "============================ ATM Snapshots ============================" << endl;
    for (auto atm : ATM::GetATMList()) {
```

```
        cout << "ATM [SN: " << atm->serial_number << ((language_option == 1) ? "] Remaining Cash: {" : "] 남은 현금: {")
            << ((language_option == 1) ? "Total: KRW " : "총 금액: ") << 50000 * atm->ATM_50000_num + 10000 * atm->ATM_10
            << ((language_option == 1) ? ", " : "원, ")
            << ((language_option == 1) ? "KRW 50000: " : "50000원: ") << atm->ATM_50000_num << ", "
            << ((language_option == 1) ? "KRW 10000: " : "10000원: ") << atm->ATM_10000_num << ", "
            << ((language_option == 1) ? "KRW 5000: " : "5000원: ") << atm->ATM_5000_num << ", "
            << ((language_option == 1) ? "KRW 1000: " : "1000원: ") << atm->ATM_1000_num << "}" << endl;
    }
    cout << "================================================================================" << endl;
    cout << "========================= Account Snapshots =========================" << endl;
    for (auto account : Account::GetAccountList()) {
        cout << ((language_option == 1) ? "Account [Bank: " : "계좌 [은행: ") << account->GetBankName()
            << ((language_option == 1) ? ", No: " : ", 번호: ") << account->GetAccountNumber()
            << ((language_option == 1) ? ", Owner: " : ", 사용자: ") << account->GetUserName()
            << ((language_option == 1) ? "] Balance: " : ", 잔액: ") << account->GetAvailableFund() << endl;
    }
    cout << "================================================================================" << endl;
}

// ================================== Main Function ==================================

int main() {
    string task_option;
    AdminCard* admin = new AdminCard();

    while (true) {
        cout << "================================ Main ================================" << endl;
        cout << "Please select a task option." << endl;
        cout << "1: Bank (Create or Use Bank)" << endl;
        cout << "2: User (Create or Use User)" << endl;
        cout << "3: ATM (Create or Use ATM)" << endl;
        cout << "/: Display all information\n" << endl;
        cout << "0: Exit Program" << endl;
        cout << "================================================================================" << endl;

        if (!(cin >> task_option)) {
            cout << "Invalid Input. Please enter a number between 0 and 4." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }

        if (!(task_option == "1" || task_option == "2" || task_option == "3" || task_option == "/" || task_option == "0"
            cout << "Invalid Input. Please try again." << endl;
            continue;
        }

        if (task_option == "0") {
            cout << "Exiting program." << endl;
            break;
        }

        // ================================== 1. Bank ==================================
        else if (task_option == "1") {
            while (true) {
                cout << "================================ Bank ================================" << endl;
                cout << "Please select an option for Bank." << endl;
                cout << "1: Create Bank" << endl;
                cout << "2: Use Bank\n" << endl;
                cout << "0: Return to Main Menu" << endl;
                cout << "================================================================================" << endl;

                int bank_option;
                if (!(cin >> bank_option)) {
                    cout << "Invalid Input. Please enter a valid number." << endl;
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    continue;
                }

                if (bank_option == 0) { break; }
```

```
                // ================================== 1. Create Bank ==================================
                if (bank_option == 1) {
                    cout << "================================ Bank ================================" << endl;
                    cout << "Enter Bank Name (0 to cancel): ";
                    string bank_name;
                    cin >> bank_name;
                    if (bank_name == "0") continue;
                    Bank* new_bank = new Bank(bank_name);
                    cout << "Bank created: " << new_bank->GetBankName() << endl;
                    cout << "Bank [" << new_bank->GetBankName() << "] created successfully!" << endl;
                    cout << "======================================================================" << endl;
                }

                // ================================== 2. Use Bank ==================================
                else if (bank_option == 2) {
                    if (Bank::GetBankList().empty()) {
                        cout << "No banks available. Please create a bank first." << endl;
                        continue;
                    }

                    cout << "================================ Bank ================================" << endl;
                    cout << "Please select the bank to use.\nAvailable Banks: " << endl;
                    for (size_t i = 0; i < Bank::GetBankList().size(); ++i) {
                        cout << i + 1 << ": " << Bank::GetBankList()[i]->GetBankName() << endl;
                    }
                    cout << "\n0: Return to Main Menu" << endl;
                    cout << "======================================================================" << endl;

                    int bank_use;
                    if (!(cin >> bank_use) || bank_use < 0 || bank_use > Bank::GetBankList().size()) {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        continue;
                    }

                    if (bank_use == 0) { break; }

                    Bank* selected_bank = Bank::GetBankList()[bank_use - 1];
                    cout << "Selected Bank: " << selected_bank->GetBankName() << endl;

                    while (true) {
                        cout << "============================== Account ==============================" << endl;
                        cout << "Would you like to create an account with the selected bank?" << endl;
                        cout << "1: Yes" << endl;
                        cout << "0: Return to Bank Menu" << endl;
                        cout << "======================================================================" << endl;

                        int account_option;
                        if (!(cin >> account_option) || (account_option != 0 && account_option != 1)) {
                            cout << "Invalid choice. Please enter 0 or 1." << endl;
                            cin.clear();
                            cin.ignore(numeric_limits<streamsize>::max(), '\n');
                            continue;
                        }

                        if (account_option == 0) { break; }

                        if (account_option == 1) {
                            cout << "============================== Account ==============================" << endl;
                            cout << "Bank Name: " << selected_bank->GetBankName() << endl;
                            cout << "Enter User Name (0 to cancel): ";
                            string user_name;
                            cin >> user_name;

                            if (user_name == "0") continue;

                            User* user = FindUser(user_name);
                            if (!user) {
                                cout << "User does not exist. Please create the user first." << endl;
                                continue;
                            }
```

```
                        cout << "Enter Account Number (12 digits, 0 to cancel): ";
                        long long int acc_num;
                        cin >> acc_num;
                        if (to_string(acc_num).length() != 12) {
                            cout << "Invalid Account Number. Please enter a 12-digit number." << endl;
                            cout << "=====================================================================" << endl
                            continue;
                        }
                        if (acc_num == 0) continue;

                        cout << "Enter Available Funds (0 to cancel): ";
                        int funds;
                        cin >> funds;
                        if (funds == 0) continue;

                        cout << "Enter Password (4 digits, 0 to cancel): ";
                        int pass;
                        cin >> pass;
                        if (to_string(pass).length() != 4) {
                            cout << "Invalid Password. Please enter a 4-digit number." << endl;
                            cout << "=====================================================================" << endl
                            continue;
                        }
                        if (pass == 0) continue;

                        try {
                            Account* new_account = new Account(selected_bank, user_name, acc_num, funds, pass);
                            cout << "Account created successfully with Bank: " << selected_bank->GetBankName() << en
                        }
                        catch (const invalid_argument& e) {
                            cout << "Error during account creation: " << e.what() << endl;
                            continue;
                        }
                    }
                }
            }
            else {
                cout << "Invalid Input. Try again." << endl;
            }
        }
    }

    // ==================================== 2. User ====================================
    else if (task_option == "2") {
        while (true) {
            cout << "================================ User ================================" << endl;
            cout << "Please select an option for User." << endl;
            cout << "1: Create User" << endl;
            cout << "2: Use User\n" << endl;
            cout << "0: Return to Main Menu" << endl;
            cout << "=====================================================================" << endl;

            int user_option;
            if (!(cin >> user_option)) {
                cout << "Invalid Input. Please enter a valid number." << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                continue;
            }

            if (user_option == 0) { break; }

            // ================================ 1. Create User ================================
            if (user_option == 1) {
                cout << "================================ User ================================" << endl;
                cout << "Enter User Name (0 to cancel): ";
                string user_name;
                cin >> user_name;
                if (user_name == "0") continue;
                User* new_user = new User(user_name);
                cout << "User [" << new_user->GetUserName() << "] created successfully!" << endl;
                cout << "=====================================================================" << endl;
```

```
            }

            // ================================== 2. Use User ====================================
            else if (user_option == 2) {
                if (User::GetUserList().empty()) {
                    cout << "No users available. Please create a user first." << endl;
                    continue;
                }

                cout << "================================ User ================================" << endl;
                cout << "Please select the user to use.\nAvailable Users: " << endl;
                for (size_t i = 0; i < User::GetUserList().size(); ++i) {
                    cout << i + 1 << ": " << User::GetUserList()[i]->GetUserName() << endl;
                }
                cout << "\n0: Return to Main Menu" << endl;
                cout << "======================================================================" << endl;

                int user_use;
                if (!(cin >> user_use) || user_use < 0 || user_use > User::GetUserList().size()) {
                    cout << "Invalid choice. Returning to Main Menu." << endl;
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    continue;
                }

                if (user_use == 0) { break; }
                User* selected_user = User::GetUserList()[user_use - 1];
                cout << "Selected User: " << selected_user->GetUserName() << endl;

                cout << "User's Accounts:\n";
                for (auto& account : Account::GetAccountList()) {
                    if (account->GetUserName() == selected_user->GetUserName()) {
                        cout << ": Bank " << account->GetBankName()
                            << ", Account No: " << account->GetAccountNumber()
                            << ", Balance: " << account->GetAvailableFund() << endl;
                    }
                }
            }
            else {
                cout << "Invalid Input. Try again." << endl;
            }
        }
    }

    // ================================== 3. ATM ====================================
    else if (task_option == "3") {
        while (true) {
            cout << "================================ ATM ================================" << endl;
            cout << "Please select an option for ATM." << endl;
            cout << "1: Create ATM" << endl;
            cout << "2: Use ATM\n" << endl;
            cout << "0: Return to Main Menu" << endl;
            cout << "======================================================================" << endl;

            int ATM_option;
            if (!(cin >> ATM_option)) {
                cout << "Invalid Input. Please enter a valid number." << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                continue;
            }

            if (ATM_option == 0) { break; }

            // ================================== 1. Create ATM  ==================================
            if (ATM_option == 1) {
                cout << "Enter Primary Bank Name (0 to cancel): ";
                string bank_name;
                cin >> bank_name;
                if (bank_name == "0") continue;

                cout << "Enter ATM Serial Number (0 to cancel): ";
```

```
                    int serial;
                    cin >> serial;
                    if (serial == 0) continue;

                    cout << "Enter ATM Type (1: Single, 2: Multi, 0 to cancel): ";
                    int type;
                    cin >> type;
                    if (type == 0) continue;

                    cout << "Enter ATM Language Type (1: Unilingual, 2: Bilingual, 0 to cancel): ";
                    int lang;
                    cin >> lang;
                    if (lang == 0) continue;

                    int num_50000, num_10000, num_5000, num_1000;
                    cout << "Enter number of KRW 50000: "; cin >> num_50000;
                    cout << "Enter number of KRW 10000: "; cin >> num_10000;
                    cout << "Enter number of KRW 5000: "; cin >> num_5000;
                    cout << "Enter number of KRW 1000: "; cin >> num_1000;

                    ATM* new_atm = new ATM(bank_name, serial, type, lang, num_1000, num_5000, num_10000, num_50000);
                    cout << "ATM [Bank: " << new_atm->GetPrimaryBankName() << ", Serial: " << new_atm->GetSerialNumber()
                }

                // ================================== 2. Use ATM ==================================
                else if (ATM_option == 2) {
                    if (ATM::GetATMList().empty()) {
                        cout << "No ATMs available. Please create an ATM first." << endl;
                        continue;
                    }
                    cout << "================================ ATM ================================" << endl;
                    cout << "Please select the ATM to use.\nAvailable ATMs: " << endl;
                    for (size_t i = 0; i < ATM::GetATMList().size(); ++i) {
                        cout << i + 1 << ": ATM [SN: " << ATM::GetATMList()[i]->GetSerialNumber()
                            << ", Bank: " << ATM::GetATMList()[i]->GetPrimaryBankName()
                            << ", Type: " << ATM::GetATMList()[i]->GetATMType() << "]" << endl;
                    }
                    cout << "\n0: Return to Main Menu" << endl;
                    cout << "=====================================================================" << endl;

                    int atm_use;

                    if (!(cin >> atm_use) || atm_use < 0 || atm_use > ATM::GetATMList().size()) {
                        cout << "Invalid choice. Returning to Main Menu." << endl;
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        continue;
                    }

                    if (atm_use == 0) { break; }

                            ATM* selected_atm = ATM::GetATMList()[atm_use - 1];
                            if (selected_atm == nullptr) {
                                cout << "Error: ATM not found!" << endl;
                                continue;
                            }

                            cout << "Selected ATM [Bank: " << selected_atm->GetPrimaryBankName() << ", SN: " << selected_atm->Ge
                            selected_atm->BeforeInterface();
                        }
                        else {
                            cout << "Invalid Input. Try again.\n";
                        }
                    }
                }

                // ============================ 4. Display all information ============================
                else if (task_option == "/") {
                    cout << "============================== Information ==============================" << endl;
                    cout << "Displaying all Banks, Users, Accounts, and ATMs: " << endl;

                    cout << "Banks:\n";
```

```
            for (auto bank : Bank::GetBankList()) {
                cout << "Bank Name: " << bank->GetBankName() << endl;
            }

            cout << "Users:\n";
            for (auto user : User::GetUserList()) {
                cout << "User Name: " << user->GetUserName() << endl;
            }

            cout << "Accounts:\n";
            for (auto account : Account::GetAccountList()) {
                cout << "Account [Bank: " << account->GetBankName()
                    << ", No: " << account->GetAccountNumber()
                    << ", User: " << account->GetUserName()
                    << "] Balance: " << account->GetAvailableFund() << endl;
            }

            cout << "ATMs:\n";
            for (auto atm : ATM::GetATMList()) {
                cout << "ATM [SN: " << atm->GetSerialNumber() << "] Remaining Cash {"
                    << "KRW 50000: " << atm->GetFiftyThousand() << ", "
                    << "KRW 10000: " << atm->GetTenThousand() << ", "
                    << "KRW 5000: " << atm->GetFiveThousand() << ", "
                    << "KRW 1000: " << atm->GetOneThousand() << "}"
                    << endl;
            }
            cout << "Cards:\n";
            for (auto card : Card::GetCardList()) {
                cout << card->ToString() << endl;
            }

            cout << "================================================================================" << endl;
        }
    }
    return 0;
}
```

8. Member Contribution Table and Note

Member	Contribution	Note
김예진	25%	REQ 3,4 구현 / 보고서 작성
김유진	25%	REQ 7~10 구현 / 보고서 작성
박은빈	25%	REQ 5,6 구현 / 보고서 작성
안서연	25%	REQ 1,2 구현 / 코드 총괄 / 보고서 작성

After the implementation of each part, our team used Notion and Github to share errors and updated codes. All students debugged the code and wrote the final report together; all students equally contributed.