객체지향프로그래밍 Term project 최종 결과제출

권우혁(202311014)

서대양(202311097)

백원중(202111085)

김준호(201811034)

1. 최종 보고서
2. 최종 cpp 파일
3. Requirement check sheet
4. Member Contribution

OOP Term Project Report: Automated Teller Machine (ATM) System Development

Authors: **Woohyuk Kwon(202311014), Daeyang Seo(202311097), Wonjung Baek(202111085), Junho Kim(201811034)**
Submission Date: 2024년 12월 01일

Introduction

This report documents the implementation of an ATM system as per the requirements specified in the project instructions. The system, which is the Term Project for the **DGIST Fall 2024 Object-Oriented Programming (OOP) course**, was developed using Object-Oriented Programming (OOP) principles in C++ and consists of ATM, Bank, and Account classes. This document explains how each requirement was implemented with supporting command-line screenshots.

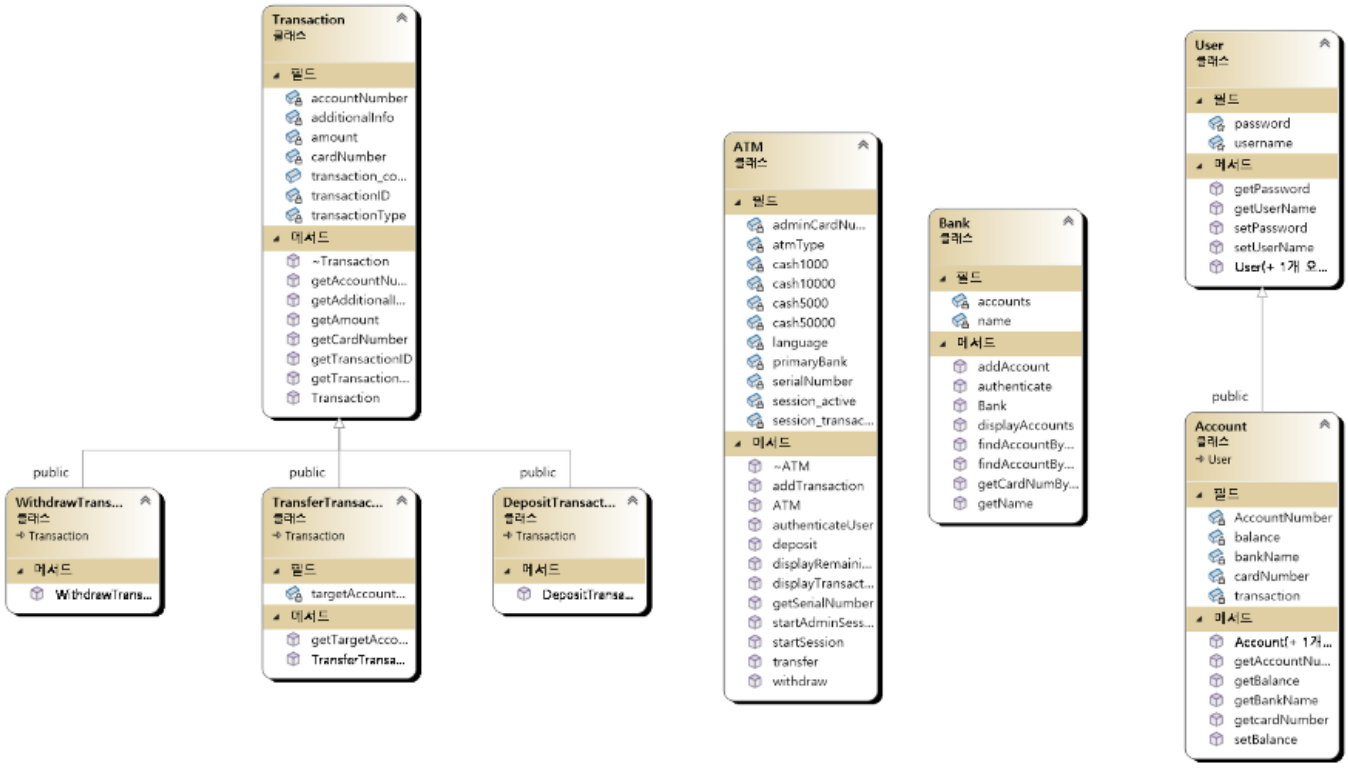


Table of Contents

        - REQ 4.6 (The deposited check does not increase available cash in the ATM)

    - REQ 5: Withdrawal

        - REQ 5.1 (An ATM shall ask a user to enter the amount of fund to withdraw)
        - REQ 5.2 (An ATM shall display an appropriate error message if there is insufficient fund in the account or insufficient cash in the ATM)
        - REQ 5.3 (Once the withdrawal is successful, the transaction must be reflected to the bank account)
        - REQ 5.4 (Some withdrawal fees may be charged)
        - REQ 5.5 (The cash withdrawal lowers available cash in the ATM)
        - REQ 5.6 (The maximum number of withdrawals per each session is 3)
        - REQ 5.7 (The maximum amount of cash withdrawal per transaction is KRW 500,000)

    - REQ 6: Transfer

        - REQ 6.1 (An ATM shall ask a user to choose the transfer types: cash transfer or account fund transfer)
        - REQ 6.2 (For both cash and account transfers, an ATM shall ask the destination account number)
        - REQ 6.3 (For cash transfer, an ATM shall ask the user to insert the cash and transaction fees)
        - REQ 6.4 (For account transfer, an ATM shall ask the source account number and the amount of fund to be transferred)
        - REQ 6.5 (Some transfer fees may be charged)
        - REQ 6.6 (The inserted cash for transfer increases available cash in the ATM)
        - REQ 6.7 (Once the transfer is successful, the transaction must be reflected to the bank account)

    - REQ 7: Transaction History

        - REQ 7.1 (When a session is started by an admin by inserting an admin card, an ATM displays a menu of “Transaction History” only)
        - REQ 7.2 (When the “Transaction History” menu is selected, an ATM displays the information of all transactions)
        - REQ 7.3 (The “Transaction History” information shall be outputted to the external file)

    - REQ 8: Multi-language Support

        - REQ 8.1 (An ATM configured with bilingual support shall provide an option for a user to choose the preferred language)
        - REQ 8.2 (Once a certain language is chosen, all menus must be displayed using the chosen language)

    - REQ 9: Exception Handling

        - REQ 9.1 (The ATM shall display an appropriate error message for each exception scenario)

    - REQ 10: Display of Account/ATM Snapshot

    - REQ 10.1 (When the character '/' is given as a console input, ATM and account snapshots are displayed)
3. List of concepts of object-oriented programming

Requirement Implementation

REQ 1: System Setup

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

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi):
```

```
Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Duplicate serial number has been detected. Please enter a unique serial number.
Serial Number(6-digit): 123
Invalid serial number. It must be exactly 6 digits.
Serial Number(6-digit):
```

ATM을 개설할 때마다 사용자로부터 Serial Number를 입력받도록 구현되었다. 또한, 각 ATM은 고유한 6자리 Serial Number를 가져야 하므로, Serial Number가 중복되면 다시 입력을 요청해야하고 입력된 Serial Number가 6자리가 아니면 올바른 형식으로 입력하라는 메시지를 출력하고 재입력을 요구해야 한다. 위 사진은 ATM1의 Serial Number를 123123으로 설정한 경우, ATM2의 Serial Number는 123123으로 설정할 수 없도록 중복이 방지되는 것을 확인할 수 있다다. 또한, Serial Number가 3자리수와 같이 올바르지 않은 경우 ATM 설정이 거부되고 재입력을 요구하는 것을 확인할 수 있다.

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

- For Single Bank ATM, the ATM is belonged to a primary bank, and only a card issued by the primary bank is considered valid.
- For a Multi-Bank ATM, there is a primary bank that manages the ATM, but a card issued by any other banks is considered valid.

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi): Single
```

ATM기를 개설하는 과정에서 Single인지, Multi인지 사용자가 선택할 수 있도록 구현하였다.

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123456
Type(Single or Multi): Single
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 234567
Type(Single or Multi): Multi
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 2 created successfully.

Would you like to create ATM 3? (yes or no) -> no
```

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 50000Won
Account [Bank: Kakao, Card Number: 2345, Username: WH2] balance: 50000Won
Account [Bank: Woori, Card Number: 3456, Username: WH3] balance: 50000Won
====================================

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 1

Please Enter your Card number: 2345
Enter Password for your account : 2345
Invalid card. Please note that this ATM is associated with Toss Bank.

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 2

Please Enter your Card number: 2345
Enter Password for your account : 2345
Access granted for Multi-Bank ATM. You are authorized to access Kakao Bank.
Authentication successful. Welcome, WH2!
```

ATM1은 Toss bank 기반 Single ATM이고, ATM2는 Toss bank 기반 Multi ATM이다. 따라서 ATM1에 Kakao 계좌인 '2345'계좌로 로그인을 할 시, Invalid card로 뜨면서 접근이 불가해진다. 반면 ATM2에 동일한 '2345' 계좌로 로그인을 할 시, 접근이 가능해지고 거래를 진행할 수 있게된다.

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

- When an ATM is configured unilingual, all information is displayed in English only.
- When an ATM is configured bilingual, a user can choose if the information is to be displayed either English or Korean (Note: if you know only one of the languages, consider using a language translation service, such as Google Translation).

Unilingual한 ATM에 로그인하는 케이스는 위의 REQ1.2의 스크린샷을 통해 확인할 수 있다. session 로그인시에 Bilingual한 ATM은 언어를 선택할지 물어보지만, Unilingual한 ATM에 로그인할시에는 그냥 영어 그대로 보여주게된다.

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi): Single
Language(Uni or Bi): Bi
```

```
Select language / 언어를 선택하세요:
1. English
2. 한국어
->
```

ATM을 설정하는 과정에서 unilingual or bilingual인지 선택할 수 있다. 사용자가 Uni를 입력했을 경우, session 로그인시에 영어로만 출력하고 Bi를 입력할 경우, session 이후에 아래 사진과 같이 영어 또는 한국어를 선택할 수 있다.

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

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi): Single
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 100
Number of initial 5,000 Cash?: 100
Number of initial 10,000 Cash?: 1023
Number of initial 50,000 Cash?: 1341
ATM 1 created successfully.
```

ATM의 초기설정에서 각 ATM의 1,000원, 5,000원, 10,000원, 50,000원 지폐 수를 설정할 수 있다.

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

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

```
======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: Toss
User Name: Kukyang
Card number (4 digits): 1234
Account Number (12 digits): 123123123123
Balance(KRW): 1000
Password: 1234
```

Bank Creation 이후, Account Creation 과정에서 계좌 개설 과정이 이루어진다. 여기서는 Bank Creation에서 생성된 은행들에 해당하는 계좌만 생성이 가능하여, Bank Name을 입력받고 이후 User Name, Account Number, Available funds(balance)와 Password들을 설정한다. Transaction History는 이후 거래가 진행되면서 그에 해당하는 정보가 입력된다.

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

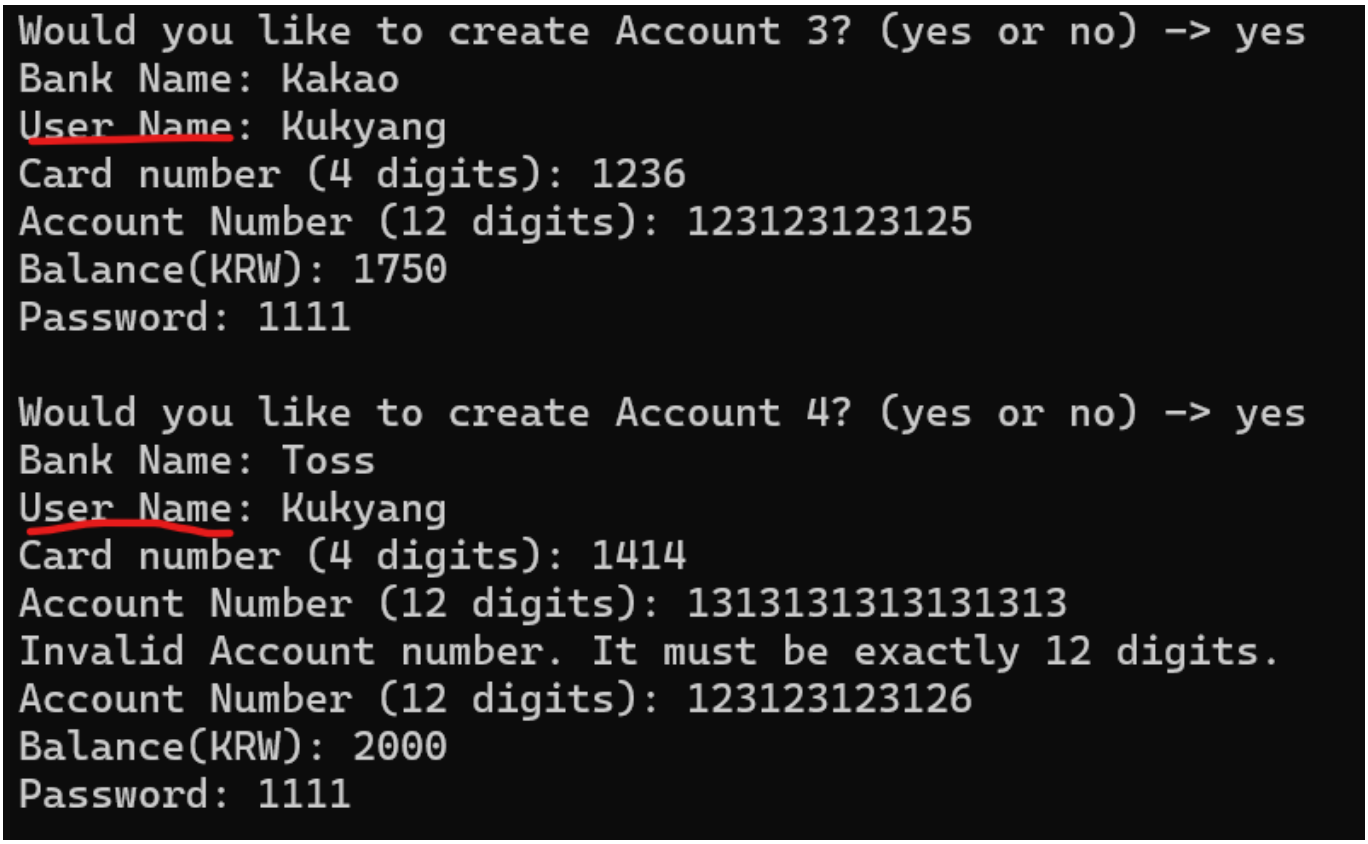
```
======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: toss
User Name: dy1
Card number (4 digits): 1234
Account Number (12 digits): 123123123123
Balance(KRW): 50000
Password: 1234

Would you like to create Account 2? (yes or no) -> yes
Bank Name: toss
User Name: dy1
Card number (4 digits): 2345
Account Number (12 digits): 321321321321
Balance(KRW): 50000
Password: 1234
```

위 사진은 dy1이름을 가진 동일한 유저가 각각 다른 toss 은행 계좌를 2개 만들 수 있다는 것을 보여준다.

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



Kukyang이라는 이름을 가진 동일한 유저가 각각 Kakao와 Toss 은행의 계좌를 개설하는 사진이다.

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

- Deposit fee for non-primary banks: KRW 2,000; the fee is paid by inserting additional cash.
- Deposit fee for primary banks: KRW 1,000; the fee is paid by inserting additional cash.
- Deposit fee for non-primary banks: KRW 2,000; the fee is paid by inserting additional cash.
- Deposit fee for primary banks: KRW 1,000; the fee is paid by inserting additional cash.
- Withdrawal fee for a primary bank: KRW 1,000; the fee is paid from the withdrawal account.
- Withdrawal fee for non-primary banks: KRW 2,000; the fee is paid from the withdrawal account.
- Account transfer fee between primary banks: KRW 2,000; the fee is paid from the source account.
- Account transfer fee between primary and non-primary banks: KRW 3,000; the fee is paid from the source account.
- Account transfer fee between non-primary banks: KRW 4,000; the fee is paid from the source account.
- Cash transfer fee to any bank type: KRW 1,000; the fee is paid by inserting additional cash.

- Account 개설 상태

```
======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: Toss
User Name: JH1
Card number (4 digits): 1234
Account Number (12 digits): 123123123123
Balance(KRW): 1000
Password: 1

Would you like to create Account 2? (yes or no) -> yes
Bank Name: Toss
User Name: JH2
Card number (4 digits): 1235
Account Number (12 digits): 123123123125
Balance(KRW): 10000
Password: 1

Would you like to create Account 3? (yes or no) -> yes
Bank Name: Kakao
User Name: JH3
Card number (4 digits): 1236
Account Number (12 digits): 123123123126
Balance(KRW): 10000
Password: 1
```

	Account 1	Account 2	Account 3
Bank	Toss	Toss	Kakao
USER	JH1	JH2	JH3
Card number	1234	1235	1236
Account number	123123123123	123123123125	123123123126

계좌 개설을 위 표와 같이 한 상태이다.

- ATM 생성 상태

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi): Single
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 123123
Number of initial 5,000 Cash?: 123123
Number of initial 10,000 Cash?: 123123
Number of initial 50,000 Cash?: 123123
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Kakao
Serial Number(6-digit): 123124
Type(Single or Multi): Multi
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 123123
Number of initial 5,000 Cash?: 123123
Number of initial 10,000 Cash?: 123123
Number of initial 50,000 Cash?: 123123
ATM 2 created successfully.
```

	ATM 1	ATM 2
Type	Single	Multi
Primary Bank	Toss	Kakao

ATM 생성을 위 표와 같이 한 상태이다.

(1) **Deposit**

- ATM 1(Toss, single) - Account 1 (Toss)

```
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
```

위 사진을 통해, Single로 설정한 Toss 뱅크에 Toss 카드를 삽입하여 입금을 시도할 경우, 1,000원의 수수료가 부과되는 것을 확인할 수 있다.

- ATM2(Kakao, Multi) - Account2(Toss)

```
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
->
```

위 사진을 통해, Multi로 설정한 Kakao 뱅크에 Toss 카드를 삽입하여 입금을 시도할 경우, 2,000원의 수수료가 부과되는 것을 확인할 수 있다.

(2) **Withdraw**

- ATM1(Toss, single) - Account1 (Toss)

```
The withdrawal amount is KRW 1000 with a fee of KRW 1000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
->
```

위 사진을 통해, Single로 설정한 Toss 뱅크에 Toss 카드를 삽입하여 출금을 시도할 경우, 1,000원의 수수료가 부과되는 것을 확인할 수 있다.

- ATM2(Kakao, Multi) - Account2(Toss)

```
The withdrawal amount is KRW 1000 with a fee of KRW 2000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
->
```

아래 사진을 통해, Single로 설정한 Toss 뱅크에 Toss 카드를 삽입하여 출금을 시도할 경우, 2,000원의 수수료가 부과되는 것을 확인할 수 있다.

(3) **Transfer**

- ATM1(Toss, Single) 보내는 계좌: Account1 (Toss) -> 받는 계좌: Account2(Toss)

```
Enter the name of the destination bank: Toss
Input destination account number: 123123123125
A transfer fee of KRW 2000 will be deducted.
1. Confirm
2. Cancel
->
```

- ATM1(Toss, Single) 보내는 계좌: Account1 (Toss) -> 받는 계좌: Account3(Kakao)

```
Enter the name of the destination bank: Kakao
Input destination account number: 123123123126
Enter the amount you wish to transfer: 1000
Your transfer fee is KRW 3000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
->
```

- ATM2(Kakao, Multi) 보내는 계좌: Account1 (Toss) -> 받는 계좌: Account2(Toss)

```
Enter the name of the destination bank: Toss
Input destination account number: 123123123125
A transfer fee of KRW 4000 will be deducted.
1. Confirm
2. Cancel
->
```

(4) **Cash Transfer**

```
Cash transfer fee costs KRW 1000. Please insert the same amount of cash to the ATM.
1. Confirm
2. Cancel
->
```

현금 계좌이체의 경우는 fee가 primary/non-primary bank 관계에 상관없이 1000원 고정이다.

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

```
Transaction Summary:
Card Number: 2345, Account Number: 2222333344444, Type: Deposit, Amount: 80000Won
Card Number: 2345, Account Number: 2222333344444, Type: Withdraw, Amount: 32000Won

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: sessio
n
Enter ATM index (1 to 2) to start session: 1

Please Enter your Card number: 0000
Admin session has been started.
Admin session started. Displaying Transaction History.
Would you like to view the Transaction History? (Enter 'yes') :yes

===== Transaction History =====
ID: 1, Card: 2345, Type: Deposit, Amount: 80000Won
ID: 2, Card: 2345, Type: Withdraw, Amount: 32000Won
Transaction history has been saved to transaction_history.txt text file.
```

앞전에서 여러 거래를 진행한 후, session을 다시 실행하여 admin card에 해당하는 '0000' 카드로 로그인하면, Transaction History를 확인할 수 있다.

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

- (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.

```
Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123456
Type(Single or Multi): Single
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 234567
Type(Single or Multi): Multi
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 2 created successfully.

Would you like to create ATM 3? (yes or no) -> no

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: /


=======< 'Account/ATM Snapshot' >=======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
```

위 사진과 같이 ATM Creation 화면에서 각 지폐별로 개수를 설정해서 Initialization을 진행할 수 있다. 이후 display Snapshot에 접근하면 남아 있는 현금의 잔여 개수가 표시되며, 각 지폐별로 잔량이 나타난다. 예를 들어, 1,000원권, 5,000원권, 10,000원권, 그리고 50,000원권의 지폐 개수가 각각 얼마 남아 있는지가 명확하게 확인 가능하다.

- ■ Therefore, all actions of inserting cash to the ATM are performed by specifying the number of bills for each denomination.
- (Check type) Any amount over KRW 100,000 check (e.g., KRW 100,000, 100,001, 234,567 are all valid checks)

<현금>

```
A fee of KRW 1,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 12
Enter the number of KRW 5,000 bills you want to deposit: 12
Enter the number of KRW 10,000 bills you want to deposit: 12
Enter the number of KRW 50,000 bills you want to deposit: 12
The total deposit amount is KRW 792000
```

<수표>

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 1
Check #1: Enter the check amount (Minimum: KRW 100,000): 100200
Deposit successful! Your new balance is KRW 756960.
Deposit completed.
```

위와같이 현금을 ATM에 넣어줄때 각 단위 지폐별로 입력받고, 수표를 넣어줄때는 10만원만 넘어가면 얼마던지 유효한 수표로 인정한다.

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

- 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.
- 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.

```
======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: Toss
User Name: JH1
Card number (4 digits): 1234
Account Number (12 digits): 123123123123
Balance(KRW): 1000
Password: 1

Would you like to create Account 2? (yes or no) -> yes
Bank Name: Toss
User Name: JH2
Card number (4 digits): 1235
Account Number (12 digits): 123123123125
Balance(KRW): 10000
Password: 1

Would you like to create Account 3? (yes or no) -> yes
Bank Name: Kakao
User Name: JH3
Card number (4 digits): 1236
Account Number (12 digits): 123123123126
Balance(KRW): 10000
Password: 1
```

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123123
Type(Single or Multi): Single
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 123123
Number of initial 5,000 Cash?: 123123
Number of initial 10,000 Cash?: 123123
Number of initial 50,000 Cash?: 123123
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Kakao
Serial Number(6-digit): 123124
```

```
Type(Single or Multi): Multi
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 123123
Number of initial 5,000 Cash?: 123123
Number of initial 10,000 Cash?: 123123
Number of initial 50,000 Cash?: 123123
ATM 2 created successfully.
```

Account 와 ATM은 compile 이후, Bank Initialization => Account Creation => ATM Creation Step 을 순서대로 거치면서 생성된다.

REQ 2: ATM Session

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

- A session begins when a user inserts a card (card number input).

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 1) to start session: 1

Please Enter your Card number: 1234
Enter Password for your account :
1
Access granted for Single-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, JH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:
```

ATM 선택 후, 유저가 카드를 입력하면서 session이 시작된다.

(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).

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:2

Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 66000
The withdrawal amount is KRW 66000 with a fee of KRW 1000.
50000: 1, 10000: 1, 5000: 1, 1000: 1
Would you like to confirm the withdrawal?
1. Confirm
2. Cancel
-> 1
Successfully withdrawn! Remaining balance: KRW 933000
ATM has run out of cash. Ending session.

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: /


======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123412] remaining cash: {KRW 50000 : 0, KRW 10000 : 0, KRW 5000 : 0, KRW 1000 : 0}
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
```

ATM(SerialNum : 123412)에서 각 지폐의 종류가 하나씩만 남은 상황에서, 66000원을 인출하면 ATM 내부의 총 지폐 개수가 0개가 된다. withdraw 직후 남아있는 현금 개수를 체크하여 'ATM has run out of cash. Exiting session' 을 출력하며 현재 session이 자동으로 종료된다.

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

- (e.g.) Account/card info, transaction types (deposit, transfer, withdrawal), and their amount, ...
- If no transactions are successfully completed during the session, it is acceptable not to print a summary.

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:4

Transaction Summary:
Card Number: 1234, Account Number: 123123123123, Type: Transfer, Amount: 164000Won, target account: 123123123122
Card Number: 1234, Account Number: 123123123123, Type: Withdraw, Amount: 100000Won
Card Number: 1234, Account Number: 123123123123, Type: Deposit, Amount: 66000Won
Card Number: 1234, Account Number: 123123123123, Type: Deposit, Amount: 1000000Won

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program:
```

4를 입력해서 Session을 종료(Exit)하면 Transaction summary가 나온다. 각 Transaction 특징에 맞게 target account 의 표시 여부가 달라진다.

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

```
===== Transaction History =====
ID: 1, Card: 1234, Type: Deposit, Amount: 66000
ID: 2, Card: 1234, Type: Withdraw, Amount: 1000
ID: 3, Card: 1234, Type: Deposit, Amount: 250000
Transaction history has been saved to transaction_history.txt text file.
```

각 거래별마다 unique한 ID를 부여하여, 모든 거래가 다른 숫자를 가지도록 제작하였다.

REQ 3: User Authorization

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

- The ATM requests and validates the user's password via communication with the Bank class.

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 1) to start session: 1

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 1

Please Enter your Card number: 123
There's no account available with that number. Please try again.

Please Enter your Card number: 123
There's no account available with that number. Please try again.

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Single-Bank ATM. You are authorized to access toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
```

Account number가 111122223333이고, card number가 1234인 account를 생성한 뒤, ATM creation 이후 session을 시작할때 user authentication 과정을 진행한다. 먼저 ATM의 언어지원에 따라 영어/한국어를 선택하고, 직후에 authentication 과정이 이루어진다. card number를 입력했을때, 해당 카드가 valid한지 확인하는 과정을 거쳐서 최종적으로 인증에 성공하게 된다. 입력을 1234가 아닌 123을 입력했을때는, 해당 account가 존재하지 않아 invalid하다는 출력을 띄워 valid 한 card number가 들어올 수 있게 했다.

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

```
======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: Toss
User Name: WH1
Card number (4 digits): 1234
Account Number (12 digits): 111122223333
Balance(KRW): 50000
Password: 1234

Would you like to create Account 2? (yes or no) -> yes
Bank Name: Woori
User Name: WH2
Card number (4 digits): 2345
Account Number (12 digits): 222233334444
Balance(KRW): 50000
Password: 2345

Would you like to create Account 3? (yes or no) -> no

======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123456
Type(Single or Multi): Single
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Woori
Serial Number(6-digit): 234567
Type(Single or Multi): Multi
Language(Uni or Bi): Bi\
Incorrect Language Type. Please enter within 'Uni' or 'Bi'.
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 2 created successfully.

Would you like to create ATM 3? (yes or no) -> no

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 1

Please Enter your Card number: 2345
Enter Password for your account : 2345
Invalid card. Please note that this ATM is associated with Toss Bank.
```

ATM1은 현재 Toss bank의 Single 타입ATM 이므로, 여기에 invalid한 Woori bank의 계좌를 통해 접근을 시도했을때 접근이 불가능해야한다. 스크린샷에서는 그 구현이 잘 이루어졌고, 추가적으로 해당 ATM이 Toss Bank와 관련되어있다고 설명해주고 있다.
또한 invalid한 카드 말고도, 아예 존재하지 않는 카드일 경우에는 다른 error message를 띄워준다. 밑 스크린샷에서, 아예 존재하지 않는 1234567 카드가 입력되면, “There’s no account available~” 에러 메세지가 출력된다.

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 1

Please Enter your Card number: 2345
Enter Password for your account : 2345
Invalid card. Please note that this ATM is associated with Toss Bank.

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 1

Please Enter your Card number: 1234567
There's no account available with that number. Please try again.

Please Enter your Card number:
```

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

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 1) to start session: 1

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 1

Please Enter your Card number: 1234
Enter Password for your account : 123
Incorrect password. Please try again (2 attempts left).
Please re-enter your password:
```

ATM에서 입력한 비밀번호를 받아 authenticateUser 함수를 통해 인증하는 과정이다. 비밀번호가 올바르면 인증 성공 메시지를 출력하고, 계좌와 연결된 은행을 식별합니다. 인증 실패 시 적절한 메시지를 출력하며, 재입력을 요청한다. 총 3번의 시도가 가능하다.

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

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 1) to start session: 1

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 1

Please Enter your Card number: 1234
Enter Password for your account : 1294786
Incorrect password. Please try again (2 attempts left).
```

카드번호 '1234'에 맞는 적절한 password는'1234'일때, '1294786'이라는 틀린 password가 들어오면 "Incorrect password~" error message를 띄우고, 해당 session내에서 남은 시도횟수를 보여준다.

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

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 1) to start session: 1

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 1

Please Enter your Card number: 1234
Enter Password for your account : 1294786
Incorrect password. Please try again (2 attempts left).
Please re-enter your password: 1245
Incorrect password. Please try again (1 attempts left).
Please re-enter your password: 1
Incorrect password. Please try again (0 attempts left).
Maximum attempts reached. Session terminated for security, and your card has been returned.

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
```

해당 card number에 맞는 비밀번호는 '1234' 일때, 1294786, 1245, 1 이렇게 3번 잘못된 비밀번호를 입력했을 때, session이 종료되고 카드가 유저에게 돌아가도록 작동한다.

REQ 4: Deposit

(REQ4.1) An ATM shall take either cash or check from a user. - The number of bills is entered separately for each denomination

- Users can deposit cash or checks into the ATM, adhering to the limit of 50 bills or 30 checks per transaction.

deposit 종류를 Cash deposit과 Check deposit으로 나눠서 구현하였다. Cash deposit을 선택하면 현금을 사용한 deposit 과정이 진행되고, Check deposit을 선택하면 수표를 사용한 deposit 과정이 시작된다.

(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.

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 2,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 50
Enter the number of KRW 5,000 bills you want to deposit: 0
Enter the number of KRW 10,000 bills you want to deposit: 0
Enter the number of KRW 50,000 bills you want to deposit: 0
Total number of bills deposited: 50
The total deposit amount is KRW 50000
.Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Deposit successful! Your new balance is KRW 708000.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:1

<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 2,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 51
Enter the number of KRW 5,000 bills you want to deposit: 0
Enter the number of KRW 10,000 bills you want to deposit: 0
Enter the number of KRW 50,000 bills you want to deposit: 0
Total number of bills deposited: 51
You cannot deposit more than 50 bills at a time.
!!Session terminated!!
```

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 2,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 40
Enter the number of KRW 5,000 bills you want to deposit: 5
Enter the number of KRW 10,000 bills you want to deposit: 5
```

```
Enter the number of KRW 50,000 bills you want to deposit: 5
Total number of bills deposited: 55
You cannot deposit more than 50 bills at a time.
!!Session terminated!!
```

Cash deposit에서, 현금을 넣는 개수가 50개가 넘어갈때 갯수 초과로 인한 에러 메세지가 출력되고 있다. 현금 개수를 적절하게 분배하더라도, 총합 개수가 50이 넘어가면 에러 메세지가 출력된다.

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000.
Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 30
Check #1: Enter the check amount (Minimum: KRW 100,000): 100000
The Check amount is KRW 100000.
```

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000.
Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 31
You cannot deposit more than 30 checks at a time.
```

Check deposit에서는, 수표를 30개까지 넣을 수 있다. 수표를 넣는 개수가 30개가 넘어간 31개부터는 갯수 초과로 인한 에러 메시지가 정상적으로 출력된다.

(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).

-3.1. Cash deposit

- ■ 3.1.1. primary bank

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 1
Enter the number of KRW 5,000 bills you want to deposit: 1
Enter the number of KRW 10,000 bills you want to deposit: 1
Enter the number of KRW 50,000 bills you want to deposit: 1
Total number of bills deposited: 4
The total deposit amount is KRW 66000
.Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Deposit successful! Your new balance is KRW 76000.
```

primary bank 이므로 fee는 1000원 이다. 수수료 입금에 동의하면서 1000원을 낸 것이다. 원래 10,000원이 들어 있었고, 수수료를 제외한 66,000원이 입금되면서 76,000원이 된다.

- ■ 3.1.2. not primary bank

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 2,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 1
Enter the number of KRW 5,000 bills you want to deposit: 1
Enter the number of KRW 10,000 bills you want to deposit: 1
Enter the number of KRW 50,000 bills you want to deposit: 1
Total number of bills deposited: 4
The total deposit amount is KRW 66000
.Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Deposit successful! Your new balance is KRW 142000.
```

Multi ATM에서 primary bank 가 아니므로 fee는 2000원이다. 수수료 입금에 동의하면서 2000원을 낸 것이다. 원래 76,000원이 들어있었고, 수수료를 제외한 66,000원이 입금되면서 142,000원이 된다.

-3.2. Check deposit`

- ■ 3.2.1. Primary bank

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:1

<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 2
Check #1: Enter the check amount (Minimum: KRW 100,000): 100000
Check #2: Enter the check amount (Minimum: KRW 100,000): 100000
Deposit successful! Your new balance is KRW 342000.
```

primary 이므로 수수료는 1000원이다. 수수료에 동의하면서 수수료를 지불한다. 원래 142000원이었는데 10만원 짜리 수표 2장을 입금하면서 342,000원이 된다.

- ■ 3.2.2. Non Primary Bank

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:1

<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 2,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 2
Check #1: Enter the check amount (Minimum: KRW 100,000): 100000
Check #2: Enter the check amount (Minimum: KRW 100,000): 100000
Deposit successful! Your new balance is KRW 542000.
```

non primary 이므로 수수료는 2000원. 수수료에 동의하면서 수수료를 지불한다. 원래 342000원이었는데 10만원 짜리 수표 2장을 입금하면서 542,000원이 된다.

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

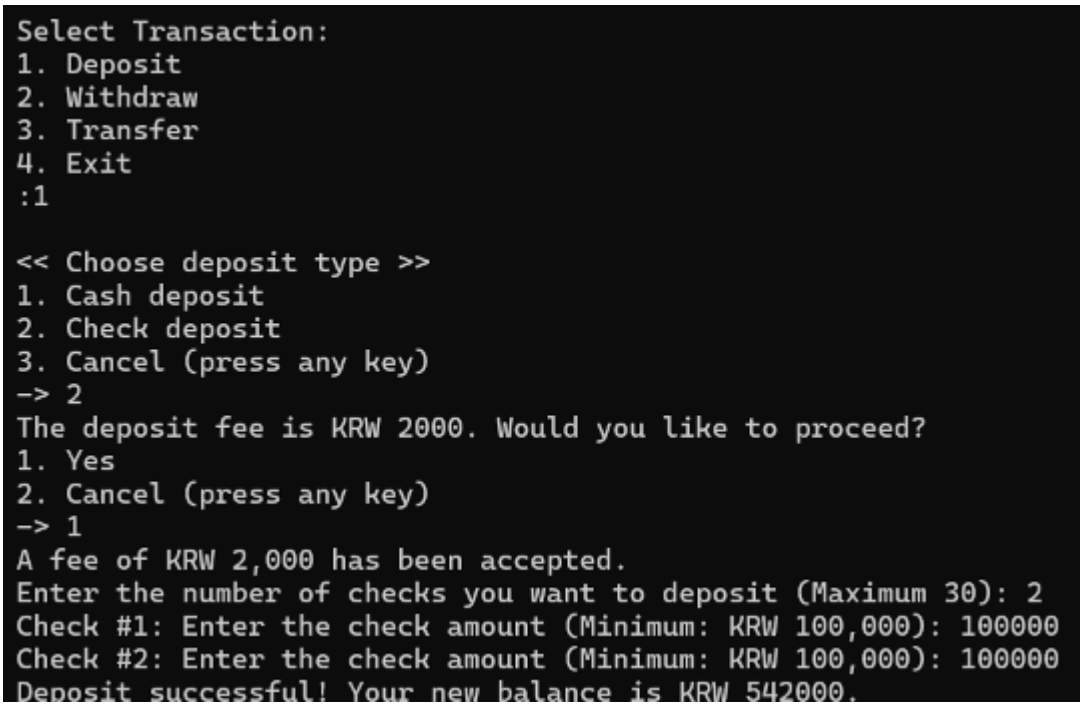
- In case of a check deposit, a fee must be served as cash.
- The deposit amount and the fee must be entered separately, with one entry for the deposit and another for the fee.
- Primary bank

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:1

<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 2
Check #1: Enter the check amount (Minimum: KRW 100,000): 100000
Check #2: Enter the check amount (Minimum: KRW 100,000): 100000
Deposit successful! Your new balance is KRW 342000.
```

Primary Bank인 경우 입금할 때, 수수료가 1000원이다. 수수료에 동의를 하면 ATM에 1000원 지폐를 수수료로 낸 것으로 처리한다(The deposit amount and the fee must be entered separately).

- Non Primary Bank



Non primary Bank인 경우 입금할 때, 수수료가 2000원이다. 수수료에 동의를 하면 ATM에 1000원 지폐를 2장으로 수수료 낸 것으로 처리한다.(The deposit amount and the fee must be entered separately).

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

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 1
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: 10
Enter the number of KRW 5,000 bills you want to deposit: 10
Enter the number of KRW 10,000 bills you want to deposit: 10
Enter the number of KRW 50,000 bills you want to deposit: 10
Total number of bills deposited: 40
The total deposit amount is KRW 660000
.Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Deposit successful! Your new balance is KRW 670000.
```

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123123] remaining cash: {KRW 50000 : 20, KRW 10000 : 20, KRW 5000 : 20, KRW 1000 : 21}
ATM [SerialNum: 123124] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: toss, Card Number: 1234, Username: toss1] balance: 670000Won
Account [Bank: toss, Card Number: 1235, Username: toss2] balance: 10000Won
Account [Bank: kakao, Card Number: 1236, Username: kakao1] balance: 10000Won
====================================

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program:
```

ATM(serial num: 123123)에 cash deposit기능으로 , primary bank 계좌에 각 현금을 10장씩 입금하는 상황(primary 수수료 1000원). 원래 ATM에 모든 현금이 10장씩 있던 상황에서, 각 현금이 다시 10장씩 입금되면서 총 20장이 된다. 여기서 천원권 이 21장인 이유는 수수료 1000원을 받았기 때문에 한장이 더 많다.

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

```
<< Choose deposit type >>
1. Cash deposit
2. Check deposit
3. Cancel (press any key)
-> 2
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of checks you want to deposit (Maximum 30): 1
Check #1: Enter the check amount (Minimum: KRW 100,000): 100000
Deposit successful! Your new balance is KRW 770000.
```

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123123] remaining cash: {KRW 50000 : 20, KRW 10000 : 20, KRW 5000 : 20, KRW 1000 : 22}
ATM [SerialNum: 123124] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: toss, Card Number: 1234, Username: toss1] balance: 770000Won
Account [Bank: toss, Card Number: 1235, Username: toss2] balance: 10000Won
Account [Bank: kakao, Card Number: 1236, Username: kakao1] balance: 10000Won
====================================
```

ATM(serial num: 123123)에 check deposit기능으로 , primary bank 계좌에 10만원 수표 한장을 입금한 상황 (primary 수수료 1000원). 수수료 천원 지불로 인해 1000원 권의 장수가 21장에서 22장이 되었고, 다른 현금의 개수는 동일하다.

REQ 5: Withdrawal

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

- 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)
- 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

```
Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 10000
The withdrawal amount is KRW 10000 with a fee of KRW 1000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
-> 1
Successfully withdrawn!
Remaining balance: KRW 759000
ATM remaining cash:
50000: 20, 10000: 19, 5000: 20, 1000: 22
```

770,000 원이 있던 계좌에서 10000원을 출금했다. 수수료 1000원은 계좌에서 인출되면서 770,000-10,000-1,000 = 759,000 원이 남는다. ATM에서는 만원권 한장이 줄어서 19장이 됐다.

여기서 추가적으로 16000원을 출금해보자

- ```
  Would you like to proceed with a withdrawal?
  1. Yes
  2. No (End withdrawal session)
  -> 1
  Enter the amount to withdraw: 16000
  The withdrawal amount is KRW 16000 with a fee of KRW 1000.
  Would you like to proceed?
  1. OK
  2. Cancel (push any key)
  -> 1
  Successfully withdrawn!
  Remaining balance: KRW 742000
  ATM remaining cash:
  50000: 20, 10000: 18, 5000: 19, 1000: 21
  ```

759,000원이었던 계좌에서 수수료 천원과 16000원이 출금되어 742,000원이 되었다. 16,000원은 만원 한장, 5천원 한장, 그리고 천원 한장이므로 각 한장씩 ATM 잔여 현금이 줄어 들었다.

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

```
ATM remaining cash:
50000: 0, 10000: 18, 5000: 19, 1000: 22

Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 500000
The ATM does not have enough cash for this withdrawal.
```

ATM에 5만원권 0장, 만원권 18장, 5천원권 19장, 천원권 22장인 상황에서 50만원 인출을 하려고 하면, ATM에 현금이 부족하다는 에러가 뜬다.

(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).

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10,
 KRW 1000 : 10}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10,
 KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 50000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 50000Won
====================================
```

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:2

Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 24000
The withdrawal amount is KRW 24000 with a fee of KRW 1000.
50000: 0, 10000: 2, 5000: 0, 1000: 4
Would you like to confirm the withdrawal?
1. Confirm
2. Cancel
-> 1
Successfully withdrawn! Remaining balance: KRW 25000
```

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 8, KRW 5000 : 10,
KRW 1000 : 6}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10,
 KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 25000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 50000Won
====================================
```

ATM(SerialNum: 123456, Toss Single ATM)에서, Account '1234'로 session을 로그인하여 withdraw과정을 진행했다. 24000원을 출금하면, primary bank 수수료는 1,000원이 계좌에서 자동으로 차감된다. 따라서, 5만원이었던 계좌에는 withdraw amount 24,000원 + fee 1,000 원까지 해서 총 25,000원이 정상적으로 빠지는 것을 확인할 수 있다.

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

(1) Withdrawal fee for a primary bank: KRW 1,000

```
Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 10000
The withdrawal amount is KRW 10000 with a fee of KRW 1000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
-> 1
Successfully withdrawn!
Remaining balance: KRW 759000
ATM remaining cash:
50000: 20, 10000: 19, 5000: 20, 1000: 22
```

Primary Bank(Atm과 카드 모두 toss인 상황) 에서 fee 1000원이 청구된다.

(2) Withdrawal fee for non-primary banks: KRW 2,000;

```
Please Enter your Card number: 1234
Enter Password for your account : 1
Access granted for Multi-Bank ATM. You are authorized to access toss Bank.
Authentication successful. Welcome, toss1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:2

Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 1000
The withdrawal amount is KRW 1000 with a fee of KRW 2000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
->
```

Non-primary bank (atm은 kakao, 카드는 toss) 일 때 fee 2000원이 청구된다.

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

```
Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 24000
The withdrawal amount is KRW 24000 with a fee of KRW 1000.
50000: 0, 10000: 2, 5000: 0, 1000: 4
Would you like to confirm the withdrawal?
1. Confirm
2. Cancel
-> 1
Successfully withdrawn! Remaining balance: KRW 25000

Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 2
Withdrawal session has ended.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:/


======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 8, KRW 5000 : 10, KRW 1000 : 6}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
```

각 종류의 지폐가 10개씩 들어있던 상황에서, 24,000원을 출금하였다. 24,000원은 5만원0장, 만원 2장, 5천원 0장, 그리고 천원 4장이므로 해당 지폐 개수에 맞게 ATM에서 잔여 지폐 개수가 줄어들었다.

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

- If a user wants to withdraw four times, it needs to end the current session after withdrawing three times and restart another session for one more withdrawal.

```
Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 1000
The withdrawal amount is KRW 1000 with a fee of KRW 2000.
Would you like to proceed?
1. OK
2. Cancel (push any key)
-> 1
Successfully withdrawn!
Remaining balance: KRW 730000
ATM remaining cash:
50000: 10, 10000: 10, 5000: 10, 1000: 7
The maximum number of withdrawals per session is 3. Please restart another session for more withdrawals.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:
```

3번 인출 후 추가적으로 withdraw를 같은 session에서 시도할 경우, 경고 메시지와 함께 withdraw가 진행되지 않고, Select Transaction 화면으로 돌아간다.

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

```
Would you like to proceed with a withdrawal?
1. Yes
2. No (End withdrawal session)
-> 1
Enter the amount to withdraw: 1000000
The maximum withdrawal amount per transaction is KRW 500000.
```

1,000,000원 인출을 시도했다. 최대 500,000원 인출이 가능하다는 경고메시지와 함께 인출이 진행되지 않는다.

REQ 6: Transfer

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

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit
t program: session
Enter ATM index (1 to 3) to start session: 1

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Single-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
->
```

session 시작 후 Transfer를 선택하면, Cash Transfer나 Account transfer 중에 어떤 거래를 시작할건지 물어보게 한다.

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

(1) Cash transfer

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 1
Enter the name of the destination bank: Toss
Input destination account number (if you want to cancel, press 0) :222233334444
A transfer fee of KRW 2000 will be deducted.
1. Confirm
2. Cancel
-> 1
```

Cash transfer에서, ATM이 destination bank를 먼저 물어보고, 해당 bank가 존재하는지 확인한 뒤에 destination account number를 입력받는다.

(2) Account transfer

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to qu
t program: session
Enter ATM index (1 to 3) to start session: 1

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Single-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Toss
Input destination account number (if you want to cancel, press 0) :222233334444
Enter the amount you wish to transfer:
```

Account transfer에서도 마찬가지로 destination bank를 물어보고, 해당 destination bank의 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.

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 1
Enter the name of the destination bank: toss
Input destination account number (if you want to cancel, press 0) :321321321321
Cash transfer fee costs KRW 1000. Please insert the same amount of cash to the ATM.
1. Confirm
2. Cancel
-> 1
Please insert the cash to complete the transfer.
Enter the number of KRW 1,000 bills: 1
Enter the number of KRW 5,000 bills: 1
Enter the number of KRW 10,000 bills: 1
Enter the number of KRW 50,000 bills: 1
The total amount entered is: 66000 KRW.
1. Confirm
2. Re-enter cash
-> 1
Successfully transferred 66000 KRW.
Successfully transferred 66000 KRW to account 321321321321.
```

Cash transfer에서, 먼저 primary/non-primary bank 에 따른 수수료를 체크해서, 사용자에게 먼저 이 수수료를 지불할건지 물어본다. 이후 Confirm을 눌러서 수수료를 지불하고, 넣을 현금의 개수를 입력하게된다. 넣은 현금의 개수를 총합한 값을 다시 사용자에게 알려주며 한 번 더 verify 과정을 거치고, (1000+5000+10000+50000=66000) 수수료를 제외한 총 넣은 지폐가 원하는 계좌에 송금된다.

(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.)

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Toss
Input destination account number (if you want to cancel, press 0) :222233334444
Enter the amount you wish to transfer: 50000
Your transfer fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Successfully transferred 50000 KRW to account 222233334444.
There is KRW 448000 in your account.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
```

Account Transfer에서는 먼저 목적지 계좌의 bank이름을 물어보고, 계좌번호를 입력받은 뒤, 계좌를 성공적으로 찾게되면 transfer할 금액을 입력한다. source Account는 이미 session 로그인할때 로그인된 계좌이므로, 해당 정보를 활용하여 source Account 정보는 다시 물어보지 않는다.

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

(1) Initial Bank/Account/ATM Setting

```
======< 'Bank Initialization' >======
Enter bank name for initialization(or type 'done' to finish): Toss
[Toss] bank has been created!
Enter bank name for initialization(or type 'done' to finish): Woori
[Woori] bank has been created!
Enter bank name for initialization(or type 'done' to finish): Kakao
[Kakao] bank has been created!
Enter bank name for initialization(or type 'done' to finish): done
Bank initialization has been completed. Proceeding to account Initializaiton.

======< 'Account Creation' >======

Would you like to create Account 1? (yes or no) -> yes
Bank Name: Toss
User Name: WH1
Card number (4 digits): 1234
Account Number (12 digits): 111122223333
Balance(KRW): 500000
Password: 1234

Would you like to create Account 2? (yes or no) -> yes
Bank Name: Toss
User Name: WH2
Card number (4 digits): 2345
Account Number (12 digits): 222233334444
Balance(KRW): 500000
Password: 2345

Would you like to create Account 3? (yes or no) -> yes
Bank Name: Woori
User Name: WH3
Card number (4 digits): 3456
Account Number (12 digits): 333344445555
Balance(KRW): 500000
Password: 3456

Would you like to create Account 4? (yes or no) -> no
```

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123456
Type(Single or Multi): Single
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Woori
Serial Number(6-digit): 234567
```

```
Serial Number(6-digit): 234567
Type(Single or Multi): Multi
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 2 created successfully.

Would you like to create ATM 3? (yes or no) -> yes
Primary Bank Name: Kakao
Serial Number(6-digit): 345678
Type(Single or Multi): Multi
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 3 created successfully.

Would you like to create ATM 4? (yes or no) -> no
```

(2) Transfer between primary banks(Fee : 2000)

```
Enter ATM index (1 to 3) to start session: 1

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Single-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Toss
Input destination account number (if you want to cancel, press 0) :222233334444
Enter the amount you wish to transfer: 5000
Your transfer fee is KRW 2000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Successfully transferred 5000 KRW to account 222233334444.
There is KRW 441000 in your account.
```

ATM1(Toss Single ATM)에 Card number가 1234인 Toss 카드로 로그인하여, destination bank를 계좌번호가 222233334444로 설정해두면 primary 계정 사이에서의 거래이므로 fee가 2000원으로 정상적으로 요구하는 것을 볼 수 있다.

(3) Transfer between primary and non-primary banks(Fee : 3000)

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 10, KRW 10000 : 15, KRW 5000 : 15, KRW 1000 : 28}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}
ATM [SerialNum: 345678] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 2345, Username: WH2] balance: 631000Won
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 433000Won
Account [Bank: Woori, Card Number: 3456, Username: WH3] balance: 510000Won
====================================

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 3) to start session: 2

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Multi-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Woori
Input destination account number (if you want to cancel, press 0) :333344445555
Enter the amount you wish to transfer: 5000
Your transfer fee is KRW 3000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Successfully transferred 5000 KRW to account 333344445555.
There is KRW 425000 in your account.
```

ATM2(Woori Multi ATM)에 Card number가 1234인 Toss 카드로 로그인하여, destination bank를 계좌번호가 333344445555로 설정해두면 non-primary 계정 사이에서의 거래이므로 fee가 3000원으로 정상적으로 요구하는 것을 볼 수 있다.

(4) Transfer between non-primary banks(Fee : 4000)

```
Enter ATM index (1 to 3) to start session: 2

Please Enter your Card number: 2345
Enter Password for your account : 2345
Access granted for Multi-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH2!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Toss
Input destination account number (if you want to cancel, press 0) :111122223333
Enter the amount you wish to transfer: 5000
Your transfer fee is KRW 4000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
->
```

ATM2를 사용하여 거래를 진행할 예정이다. 현재 ATM2는 Woori Bank - Multi Bank로 설정되어 있으며, 세션 로그인은 카드 번호 2345인 Toss 계좌로 진행되었다. 송금 대상 은행은 Toss Non-Primary Bank로, 계좌 번호는 111122223333입니다. 두 계좌 모두 Non-Primary Bank이므로, 송금 수수료는 4,000원이 정상적으로 부과된다.

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

```
======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 12, KRW 10000 : 13, KRW 5000 : 14, KRW 1000 : 16}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 50000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 204000Won
====================================

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 1
Enter the name of the destination bank: Woori
Input destination account number (if you want to cancel, press 0) :222233334444
Cash transfer fee costs KRW 1000. Please insert the same amount of cash to the ATM.
1. Confirm
2. Cancel
-> 1
Please insert the cash to complete the transfer.
Enter the number of KRW 1,000 bills: 2
Enter the number of KRW 5,000 bills: 2
Enter the number of KRW 10,000 bills: 2
Enter the number of KRW 50,000 bills: 2
The total amount entered is: 132000 KRW.
1. Confirm
2. Re-enter cash
-> 1
Successfully transferred 131000 KRW.
Successfully transferred 131000 KRW to account 222233334444.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:/


======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 14, KRW 10000 : 15, KRW 5000 : 16, KRW 1000 : 19}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 50000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 335000Won
```

Cash transfer에서 계좌이체할 현금을 입력하면, 수수료 현금 1000원을 포함한 모든 현금들이 ATM의 available cash 개수에 반영된다. ATM의 기존 현금은 다음과 같았다 => 1000원 : 16개, 5000원 : 14개, 10000원 : 13개, 50000 : 12개 이후에 Cash transfer에서 1000원권 2개, 5000원권 2개, 10000원권 2개, 50000원권 2개를 투입하면, 현금 수수료 1000원 1개를 포함해서 최종적으로 1000원 : 19개, 5000원 : 16개, 10000원 : 15개, 50000 : 14개 이렇게 ATM의 available cash 개수가 정상적으로 늘어나게 된다.

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

바로 위의 REQ6.6에서, Cash deposit일때 구현된 경우를 확인할 수 있다. transfer 이전의 목적지 계좌의 잔액은 : 204,000원이었고, cash trasnfer로 131,000원을 transfer하면, 최종적으로 목적지 계좌는 335,000원이 된다.

```
[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 14, KRW 10000 : 15, KRW 5000 : 16, KRW 1000 : 19}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 50000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 335000Won
==================================

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Woori
Input destination account number (if you want to cancel, press 0) :2222333344444
Enter the amount you wish to transfer: 50000
Insufficient funds in your account. Transfer canceled.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:3

<< Choose transfer type >>
1. Cash transfer
2. Account transfer
3. Cancel (press any key)
-> 2
Enter the name of the destination bank: Woori
Input destination account number (if you want to cancel, press 0) :2222333344444
Enter the amount you wish to transfer: 30000
Your transfer fee is KRW 3000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
Successfully transferred 30000 KRW to account 2222333344444.
There is KRW 17000 in your account.

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:/


======< 'Account/ATM Snapshot' >=======

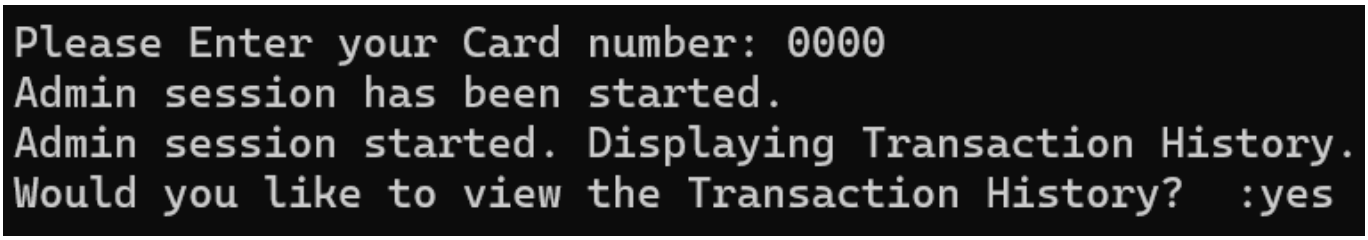
[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 14, KRW 10000 : 15, KRW 5000 : 16, KRW 1000 : 19}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 17000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 365000Won
```

Account Transfer의 경우에는, 원천 계좌의 잔액이 50000원이었고, 여기서 30,000만큼 transfer하면 fee 3,000원을 합한 33,000원이 원천계좌에서 빠져 17,000원이 남았다. 그리고 목적지 계좌에는 30,000원 만큼 transfer되었을 것이므로, 335,000원에서 30,000원 추가한 365,000원이 목적지 계좌에 남았다.

REQ 7: Transaction History

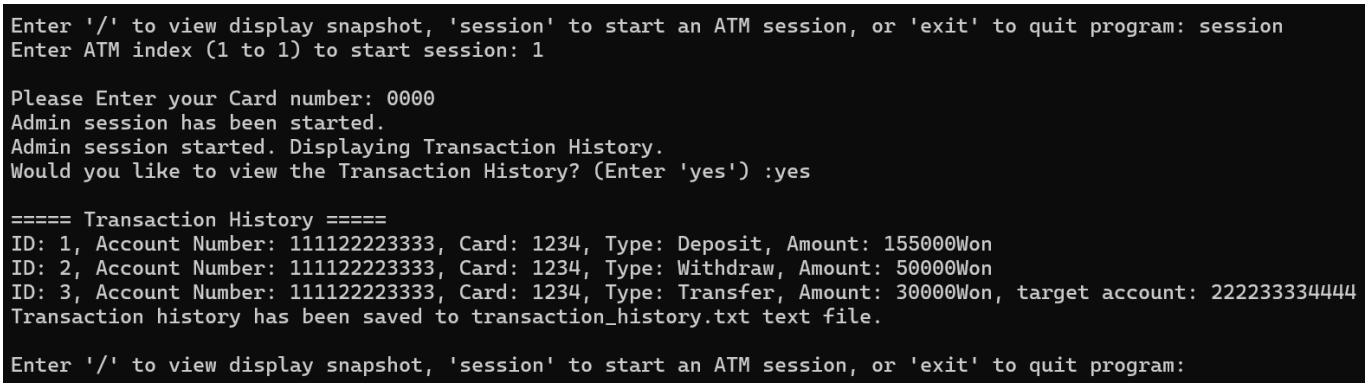
(REQ7.1) When a session is started by an admin by inserting an admin card (See REQ in System Setup), an ATM displays a menu of “Transaction History” only.



0000을 입력하면 (admin code) Admin mode로 들어간다. Transaction history를 묻고, 여기서 yes라고 하면 History를 조회할 수 있다.

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

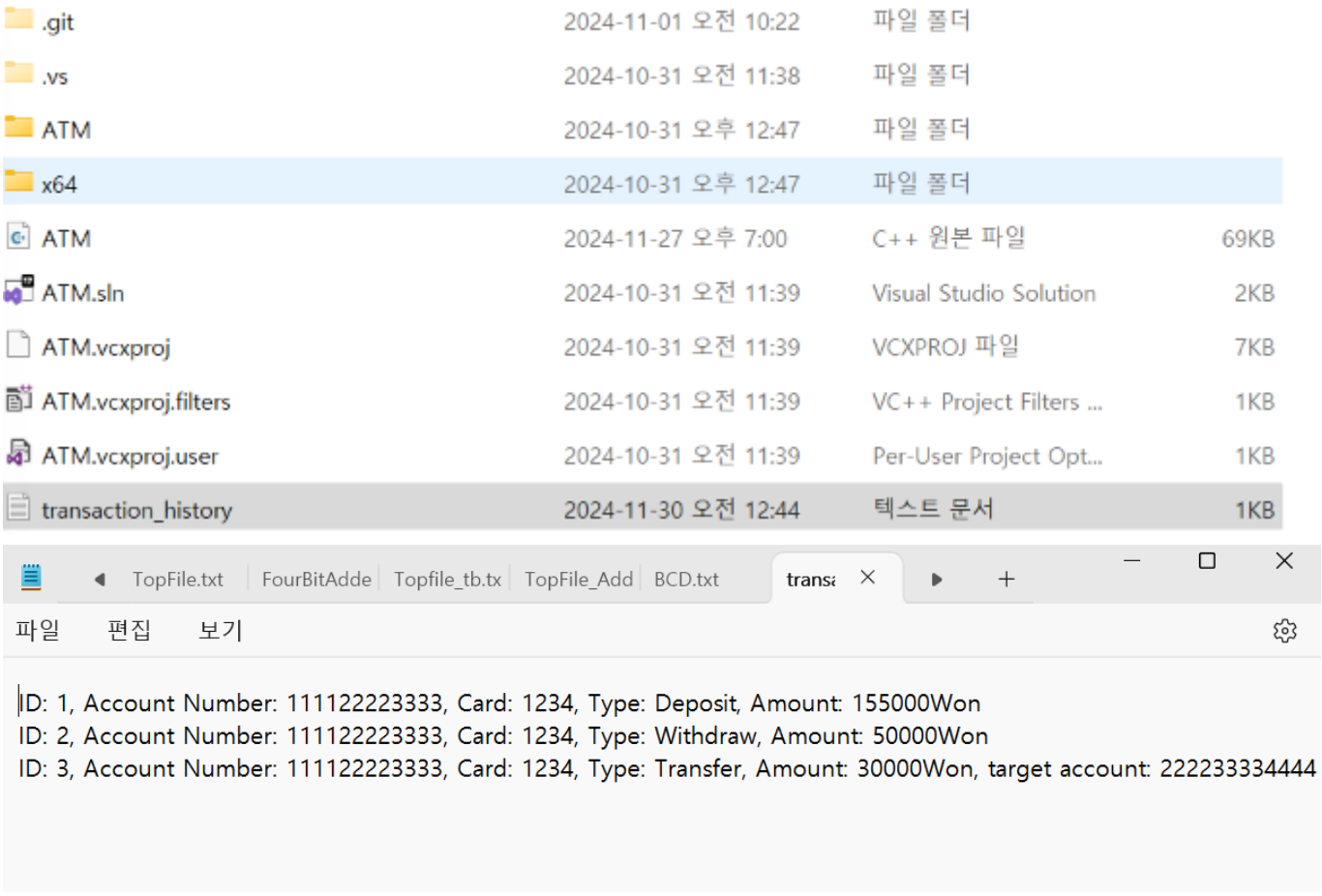
- Transaction ID, Card Number, Transaction Types, Amount, other transaction-specific information
- Each transaction may have different types of information, so they need to be appropriately displayed (e.g., a deposit transaction does not have the source account information in a transfer transaction).



yes를 입력하면 Transaction History가 뜨는데, Transaction ID가 부여되고, 순서대로 Account number(source), Card Number, Transaction Type, Amount가 표시되는 것을 확인할 수 있고, 이 Transaction History는 프로그램 전체에서의 거래 내역에 해당한다. Account Transfer와 같은 경우는 목적지 계좌에 대한 정보가 필요하므로, 기본 정보도 그대로 보여주면서 목적지 계좌도 추가해주었다.

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

The transaction history is saved to a text file (transaction_history.txt).



cpp 메인 코드가 저장된 라이브러리와 동일한 위치에 transaction_history.txt 파일을 생성하여 cmd화면에 출력된 텍스트와 동일한 텍스트가 파일에 저장된다.

REQ 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.

```
======< 'ATM Creation Step' >======

Would you like to create ATM 1? (yes or no) -> yes
Primary Bank Name: Toss
Serial Number(6-digit): 123456
Type(Single or Multi): Single
Language(Uni or Bi): Uni
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 1 created successfully.

Would you like to create ATM 2? (yes or no) -> yes
Primary Bank Name: Woori
Serial Number(6-digit): 234567
Type(Single or Multi): Multi
Language(Uni or Bi): Bi\
Incorrect Language Type. Please enter within 'Uni' or 'Bi'.
Language(Uni or Bi): Bi
Number of initial 1,000 Cash?: 10
Number of initial 5,000 Cash?: 10
Number of initial 10,000 Cash?: 10
Number of initial 50,000 Cash?: 10
ATM 2 created successfully.

Would you like to create ATM 3? (yes or no) -> no
```

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 2

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 2

카드 번호를 입력해주세요: 1234
비밀번호를 입력하세요: 1234
다중 은행 ATM에 대한 접근 권한이 부여되었습니다. Toss 은행에 접근할 수 있습니다.
인증 성공. 환영합니다, WH1!

거래를 선택하세요:
1. 입금
2. 출금
3. 송금
4. 종료
:
```

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
Enter ATM index (1 to 2) to start session: 2

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 1

Please Enter your Card number: 1234
Enter Password for your account : 1234
Access granted for Multi-Bank ATM. You are authorized to access Toss Bank.
Authentication successful. Welcome, WH1!

Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
```

여기에서 ATM2가 Bi-language ATM으로 세팅된 상태이고, 세션을 시작하고 거래에 사용할 ATM을 ATM2로 골랐을때, 언어 선택창이 먼저 띄워지고, 1번을 선택하면 이후의 문자자가 모두 영어로 출력되는 반면 2번을 선택하면 한국어로 모든 문자가 출력되고있다.

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

```
Enter ATM index (1 to 2) to start session: 2

Select language / 언어를 선택하세요:
1. English
2. 한국어
-> 2

카드 번호를 입력해주세요: 1234
비밀번호를 입력하세요: 1234
다중 은행 ATM에 대한 접근 권한이 부여되었습니다. Toss 은행에 접근할 수 있습니다.
인증 성공. 환영합니다, WH1!

거래를 선택하세요:
1. 입금
2. 출금
3. 송금
4. 종료
:1

<< 입금 유형 선택 >>
1. 현금 입금
2. 수표 입금
3. 취소 (아무 키나 누르세요)
-> 1
입금 수수료는 2000원입니다. 수수료를 지불하시겠습니까?
1. 예
2. 취소 (아무 키나 누르세요)
-> 1
2,000원의 수수료가 처리되었습니다.
천 원권 몇 장을 입금하시겠습니까? 10
오천 원권 몇 장을 입금하시겠습니까? 10
만원권 몇 장을 입금하시겠습니까? 10
오만 원권 몇 장을 입금하시겠습니까? 10
총 입금한 지폐 수: 40
입금 금액은 660000원입니다.
계속 진행하시겠습니까?
1. 예
2. 취소 (아무 키나 누르세요)
-> 1
입금이 완료되었습니다!
계좌 잔액: 710000원

거래를 선택하세요:
```

```
1. 입금
2. 출금
3. 송금
4. 종료
:2

출금 계속 진행하시겠습니까?
1. 네
2. 아니 (출금 세션을 종료)
-> 1
출금할 금액을 입력하시오: 100000
출금 금액은 100000원이며, 수수료는 2000원입니다.
계속 진행하시겠습니까?
1. 예
2. 취소 (아무 키나 누르세요)
-> 1
출금 성공!
계좌 잔액: KRW 608000

출금 계속 진행하시겠습니까?
1. 네
2. 아니 (출금 세션을 종료)
```

언어를 한국어로 선택하고, session을 시작해서 여러가지 Transaction을 진행해보았다.

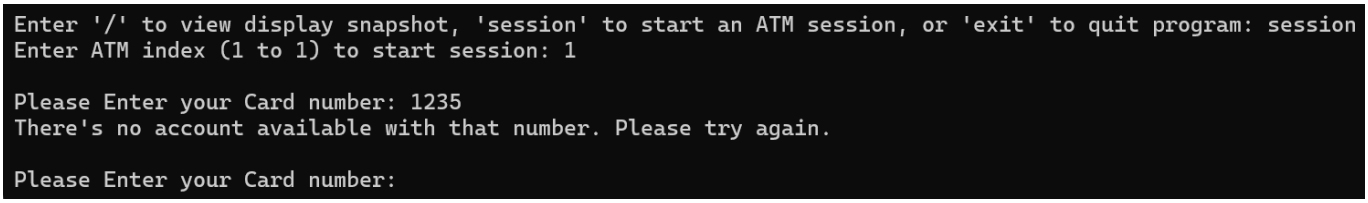
REQ 9: Exception Handling

(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.

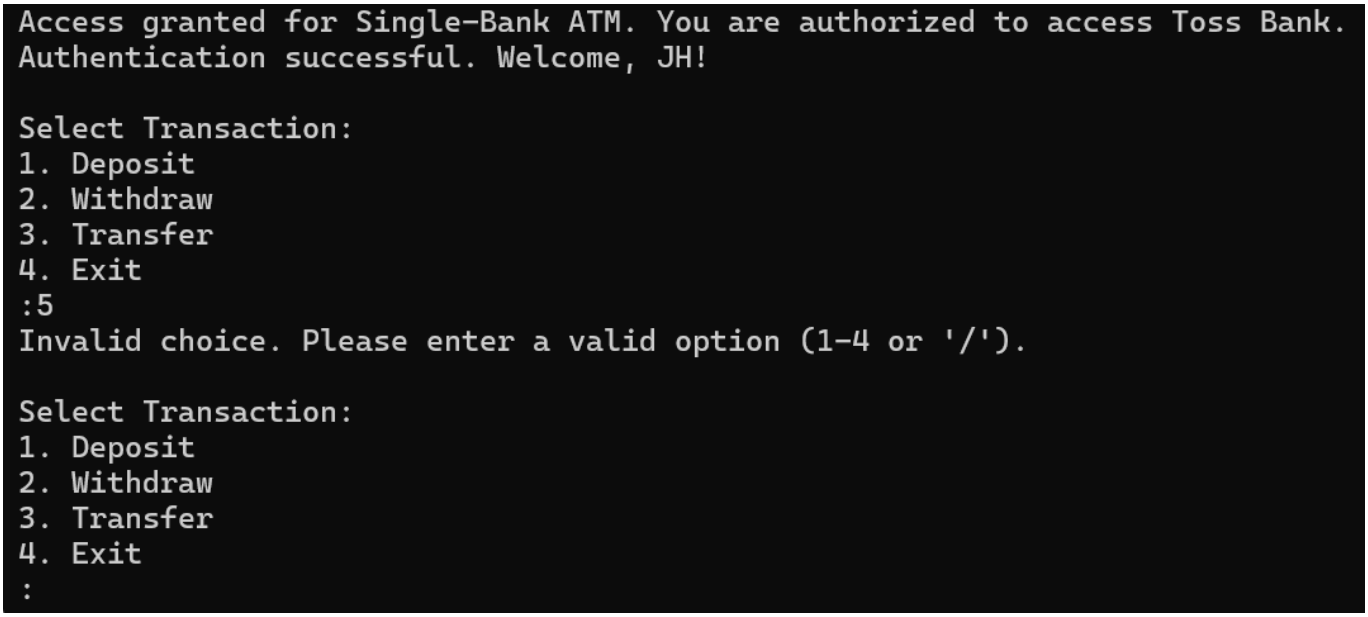
(1) 없는 ATM을 사용하겠다고 입력할시 (ex. ATM은 2대인데 3번째 ATM을 쓴다고 입력)

Invalid ATM index 라는 말과 함께 다시 session을 시작하는 지점으로 돌아간다.

(2) 없는 카드번호 입력시



(3) Session에서 없는 Transaction 번호 호출시



(4) 돈의 수량을 음수(negative)로 입력했을 때

```
The deposit fee is KRW 1000. Would you like to proceed?
1. Yes
2. Cancel (press any key)
-> 1
A fee of KRW 1,000 has been accepted.
Enter the number of KRW 1,000 bills you want to deposit: -1
Invalid input. Please enter a non-negative integer.
Enter the number of KRW 1,000 bills you want to deposit:
```

(5) Session을 입력하는데 잘못된 문자를 입력할 때

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: sassion

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program:
```

(6) Try-catch

```
    try {
        for (const ATM& atm : atms) {
            if (atm.getSerialNumber() == serialNumber) {
                throw runtime_error("Duplicate serial number detected.");
            }
        }
        isDuplicate = false;
    }
    catch (const runtime_error& e) {
        cout << "Error: " << e.what() << endl;
        isDuplicate = true;
        continue; // 다시 입력
    }
} while (isDuplicate);
```

atm serial number를 입력받을 때 이미 있는 serial number라면 runtime_error를 throw한다. catch에서서 이미 있는 serial number임을 알리는 error 메시지를 출력하고 다시 while문을 돌아 입력 받게된다.

```
while (true) {
    try {
        cout << "Balance(KRW): ";
        cin >> balance;

        if (cin.fail()) { // 잘못된 입력 처리
            throw runtime_error("Invalid input. Please enter a positive integer.");
        }

        if (balance < 0) { // 음수 처리
            throw runtime_error("Balance cannot be negative.");
        }
    }
    catch (const runtime_error& e) {
        cin.clear();
        cin.ignore(1000, '₩n');
        cout << "Error: " << e.what() << endl;
        continue; // 다시 입력 받기
    }

    // 조건 검사: 양의 정수인지 확인
    if (balance >= 0) {
        break; // 조건을 만족하면 반복 종료
    }
}
```

Balance를 입력받을 때 int가 아닌 문자를 입력받으면 if(cin.fail())에 걸려 runtime_error를 throw하고 이에 맞는 에러메시지를 출력한다. balance가 음수라면 이에 맞는 에러를 throw하고 catch문에서 음수를 입력받을 수 없다는 것을 출력한다. 이렇게 catch문에 걸리면 continue가 되서 다시 입력받게 된다.

REQ 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. - 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}) - 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)

(1) session 입력란에서 "/" 를 입력하면 모든 ATM에 대한 Snapshot이 뜬다.

```
Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: /


======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123123] remaining cash: {KRW 50000 : 123, KRW 10000 : 123, KRW 5000 : 123, KRW 1000 : 123}
ATM [SerialNum: 123122] remaining cash: {KRW 50000 : 123, KRW 10000 : 123, KRW 5000 : 123, KRW 1000 : 123}

[Account Information - Remaining Balance]
Account [Bank: toss, Card Number: 1234, Username: jh1] balance: 10000Won
Account [Bank: kakao, Card Number: 1231, Username: jh2] balance: 100000Won
====================================

Enter '/' to view display snapshot, 'session' to start an ATM session, or 'exit' to quit program: session
```

(2) Transaction 선택란에서 "/" 를 입력해도 모든 ATM에 대한 Snapshot이 뜬다.

```
Select Transaction:
1. Deposit
2. Withdraw
3. Transfer
4. Exit
:/


======< 'Account/ATM Snapshot' >======

[ATM Information - Remaining Cash]
ATM [SerialNum: 123456] remaining cash: {KRW 50000 : 11, KRW 10000 : 13, KRW 5000 : 14, KRW 1000 : 16}
ATM [SerialNum: 234567] remaining cash: {KRW 50000 : 10, KRW 10000 : 10, KRW 5000 : 10, KRW 1000 : 10}

[Account Information - Remaining Balance]
Account [Bank: Toss, Card Number: 1234, Username: WH1] balance: 121000Won
Account [Bank: Woori, Card Number: 2345, Username: WH2] balance: 80000Won
====================================
```

3. List of concepts of object-oriented programming

Encapsulation

Encapsulation은 객체지향 프로그래밍(Object Oriented Programming, OOP)의 핵심 개념 중 하나로, 데이터와 메서드를 하나의 논리적인 단위로 묶고, 객체의 내부 데이터를 보호하는 방식이다. Encapsulation을 사용하면 객체 내부의 세부 구현을 외부에서 접근할 수 없도록 차단하여 데이터의 무결성을 유지하고, 외부에서는 필요한 메서드를 통해서만 간접적으로 접근할 수 있도록 한다.

Encapsulation의 주요 목적은 데이터를 보호하고, 모듈화를 통해 유지보수성과 안정성을 높이는 데 있다. 이를 통해 객체 내부의 데이터가 무분별하게 변경되는 것을 방지하며, 외부와 객체 간의 상호작용을 간단하고 명확하게 정의할 수 있다.

Encapsulation의 주요 목적

1. **데이터 보호**: 객체의 내부 데이터를 보호하여 불필요한 접근이나 변경을 방지한다.
2. **모듈화**: 객체의 구현 세부사항을 숨기고, 객체 간의 상호작용을 단순화한다.
3. **유지보수성 향상**: 내부 구현 변경 시 외부 코드에 영향을 최소화할 수 있다.

코드에서 Encapsulation이 적용된 사례

User 클래스:

- `username`과 `password`는 `protected`로 선언하여 클래스 내부와 상속받은 클래스에서만 접근할 수 있도록 제한하였다.

- 외부에서는 `getUserName()`과 `setPassword()` 메서드를 통해 간접적으로 접근할 수 있다.
- 이를 통해 민감한 데이터를 보호하고, 필요할 경우 데이터를 검증하거나 접근 방식을 제어할 수 있다.

`Transaction` **클래스**:

- `cardNumber`, `accountNumber`와 같은 민감한 정보는 `private`로 선언하여 외부에서 직접 접근할 수 없도록 하였다.
- 대신 `getCardNumber()`, `getAccountNumber()`와 같은 `getter` 메서드를 제공하여 필요한 경우에만 데이터를 반환하도록 설계하였다.
- 이처럼 직접 접근을 차단하고 메서드를 통해 데이터를 제어함으로써, 거래 정보의 보안성을 유지할 수 있다.

`Bank` **클래스**:

- `accounts` 변수는 `private`로 선언하여 은행의 계좌 목록을 외부에서 접근하거나 수정할 수 없도록 하였다.
- 계좌와 관련된 모든 작업은 `addAccount()`와 `authenticate()`와 같은 메서드를 통해 이루어진다.
- 이를 통해 은행의 계좌 데이터가 외부의 잘못된 접근으로부터 보호되며, 데이터의 일관성을 유지할 수 있다.

`ATM` **클래스**:

- `serialNumber`, `atmType`, `cash1000` 등의 변수는 `private`로 선언하여 `ATM` 내부의 데이터를 외부에서 직접 접근하지 못하도록 하였다.
- 입금, 출금, 송금 등의 작업은 `deposit()`, `withdraw()`, `transfer()` 메서드를 통해서만 처리하도록 설계하였다.
- 이렇게 내부 데이터를 은닉하고 메서드를 통해 접근을 제한함으로써 `ATM`의 상태와 동작을 안전하게 관리할 수 있다.

Encapsulation은 단순히 데이터를 숨기는 것뿐만 아니라 객체가 제공하는 메서드를 통해 외부와의 상호작용을 정의하는 데 있다. 예를 들어, Bank 클래스의 `authenticate()` 메서드는 계좌 인증을 처리하며, 외부에서 직접 계좌 정보를 조작하지 못하게 한다. Encapsulation을 사용함으로써 클래스 간의 결합도를 낮추고, 코드의 유지보수성을 높일 수 있다. 예를 들어, `Transaction` 클래스는 다양한 거래 유형(DepositTransaction, WithdrawTransaction, TransferTransaction)을 상속받아 구현되었으며, 각 클래스는 Transaction 클래스의 메서드와 데이터를 상속받아 캡슐화를 유지한다.

코드에서 Encapsulation 적용 예시

`User` 클래스

```
class User {
protected:
    string username;
    string password;

public:
    User(const string& username, const string& password)
        : username(username), password(password) {}

    string getUserName() const { return username; }
    void setPassword(const string& password) { this->password = password; }
};
```

`Transaction` 클래스

```
class Transaction {
private:
    string cardNumber;
    int transactionID;

public:
    Transaction(int id, const string& card)
        : transactionID(id), cardNumber(card) {}

    string getCardNumber() const { return cardNumber; }
    int getTransactionID() const { return transactionID; }
};
```

Bank 클래스

```
class Bank {
private:
    unordered_map<string, Account> accounts;

public:
    void addAccount(const Account& account) {
        accounts[account.getcardNumber()] = account;
    }

    Account* authenticate(const string& cardNumber, const string& password) {
        if (accounts.count(cardNumber) && accounts[cardNumber].getPassword() ==
password) {
            return &accounts[cardNumber];
        }
        return nullptr;
    }
};
```

Abstraction

Abstraction이란 Object-Oriented Programming(OOP)에서 모듈을 설계할 때, 모듈 사용자들이 세부 구현 과정을 알 필요 없이 기능을 사용할 수 있도록 만드는 개념이다. 이를 통해 객체 간의 복잡한 관계를 단순화하고, 필요한 정보와 기능만을 제공한다.

Abstraction의 주요 목적

1. **복잡성 감소**: 불필요한 세부 사항을 숨기고, 필요한 정보와 기능만 제공한다.
2. **유지보수성 향상**: 내부 구현이 변경되더라도 외부 인터페이스를 수정하지 않음으로써 외부 코드에 미치는 영향을 최소화한다.
3. **재사용성 증대**: 공통된 인터페이스를 제공함으로써 다양한 객체나 기능을 쉽게 통합하고 재사용할 수 있다.

코드에서 Abstraction이 적용된 사례

1. ATM 클래스의 `startSession()` 함수

- 이 함수는 사용자 세션을 시작하고, 인증 및 거래를 처리하는 복잡한 로직을 내부적으로 수행한다.
- 호출자는 함수의 내부 구현을 몰라도, 세션을 시작하고 거래를 선택할 수 있다.

구현 예시

```
void startSession(const vector<Bank*>& allBanks) {
    cout << "Please Enter your Card number: ";
    string cardNumber, password;
    cin >> cardNumber;

    cout << "Enter Password: ";
    cin >> password;

    if (authenticateUser(allBanks, cardNumber, password)) {
        cout << "Session started. Welcome!\n";
        // 세부 구현은 내부적으로 처리됨
    } else {
        cout << "Authentication failed. Session terminated.\n";
    }
}
```

2. Display Transaction History

이 함수는 모든 거래 내역을 출력하거나 파일에 저장한다. 호출자는 거래 내역이 어떻게 정리되고 저장되는지 알 필요 없이, 함수 호출만으로 작업을 처리할 수 있다.

구현 예시

```
void displayTransactionHistory() {
    for (const auto& transaction : allTransactions) {
        cout << "Transaction ID: " << transaction->getTransactionID()
             << ", Card Number: " << transaction->getCardNumber()
             << ", Amount: " << transaction->getAmount() << endl;
    }
}
```

3. 다양한 클래스에서 제공하는 메서드

3.1 Bank 클래스의 `authenticate()` 메서드

이 메서드는 카드 번호와 비밀번호를 기반으로 계좌 인증을 처리한다. 호출자는 인증 과정의 세부사항을 몰라도, 함수 호출만으로 계좌를 인증할 수 있다.

구현 예시

```
Account* authenticate(const string& cardNumber, const string& password) {
    if (accounts.count(cardNumber) && accounts[cardNumber].getPassword() ==
password) {
        return &accounts[cardNumber];
    }
    return nullptr;
}
```

3.2 Transaction 클래스의 `processTransaction()` 메서드

다형성을 통해 각 거래 유형(입금, 출금, 송금)을 처리한다. 호출자는 거래의 구체적인 처리 방식을 알 필요 없이 동일한 메서드를 호출하여 작업을 수행할 수 있다.

Abstraction의 역할

객체 간 상호작용 단순화:

예: ATM과 Bank는 `authenticate()`나 `findAccountByCardNumber()`와 같은 메서드를 통해 간접적으로 상호작용한다. 이를 통해 ATM과 Bank 간의 결합도를 낮추고, 유지보수성을 높인다.

클래스 간 역할 분리:

- ATM 클래스: 거래를 처리하는 역할.
- Bank 클래스: 계좌 정보 관리.
- Transaction 클래스: 거래 세부 정보 관리.

재사용성 강화:

Bank 클래스의 계좌 검색 메서드는 ATM뿐만 아니라 관리자 세션에서도 동일하게 사용된다.

Inheritance

Inheritance는 객체지향 프로그래밍(OOP)의 핵심 개념 중 하나로, 클래스 간 계층 구조를 통해 코드 재사용성을 높이고, 유사한 클래스 간의 관계를 명확히 한다. 이를 통해 상위 클래스의 멤버 변수와 메서드를 하위 클래스에서 물려받아 사용할 수 있으며, 필요에 따라 이를 확장하거나 재정의할 수 있다.

Inheritance의 주요 특징

1. **코드 재사용성**: 상위 클래스의 멤버를 하위 클래스에서 재사용하여 중복 코드를 줄이고, 유지보수성을 높인다.
2. **계층 구조**: 클래스 간의 관계를 계층적으로 정의하여 설계를 직관적이고 체계적으로 관리할 수 있다.
3. **확장 가능성**: 하위 클래스는 상위 클래스의 동작을 기반으로 새로운 기능을 추가하거나 기존 동작을 재정의할 수 있다.

코드에서 Inheritance 적용 사례

1. Account 클래스와 User 클래스

- **Account** 클래스는 **User** 클래스를 상속받아 사용자의 정보를 관리한다.
- **User** 클래스는 `username`과 `password`를 정의하며, 이를 상속받은 **Account** 클래스는 은행 관련 데이터를 추가로 관리한다.

구현 예시

```
class User {
protected:
    string username;
    string password;

public:
    User(const string& username, const string& password)
        : username(username), password(password) {}
};

class Account : public User {
private:
    string bankName;
    string cardNumber;

public:
    Account(const string& bankName, const string& username, const string&
password)
        : User(username, password), bankName(bankName) {}
};
```

2. Transaction 클래스와 하위 클래스

Transaction 클래스는 기본적인 거래 정보를 관리하는 상위 클래스로 정의되며, 입금, 출금, 송금과 같은 특정 거래 유형은 하위 클래스에서 처리한다. 상위 클래스의 공통 속성과 동작을 재사용하며, 하위 클래스에서 필요한 부분만 확장하여 구현한다.

구현 예시

```
class Transaction {
protected:
    int transactionID;
    string cardNumber;

public:
    Transaction(int id, const string& card)
        : transactionID(id), cardNumber(card) {}
```

```
    virtual void process() const {
        cout << "Processing transaction ID: " << transactionID << endl;
    }
};

class DepositTransaction : public Transaction {
public:
    DepositTransaction(int id, const string& card)
        : Transaction(id, card) {}

    void process() const override {
        cout << "Processing deposit transaction ID: " << transactionID << endl;
    }
};

class WithdrawTransaction : public Transaction {
public:
    WithdrawTransaction(int id, const string& card)
        : Transaction(id, card) {}

    void process() const override {
        cout << "Processing withdrawal transaction ID: " << transactionID << endl;
    }
};
```

Polymorphism

Polymorphism은 하나의 인터페이스를 통해 서로 다른 타입의 객체를 다룰 수 있도록 하는 OOP의 핵심 개념이다. 이를 통해 같은 이름의 메서드나 연산자가 다양한 방식으로 동작하도록 설계할 수 있다.

Polymorphism의 주요 특징

- 인터페이스 통합: 다양한 타입의 객체를 하나의 인터페이스로 통합하여 다룰 수 있다.
- 유연성 증가: 구체적인 객체 타입에 의존하지 않고, 상위 클래스나 인터페이스에 의존하여 일반화된 동작을 제공한다.
- 확장성 강화: 새로운 타입이 추가되어도 기존 코드를 최소한으로 수정하거나 수정 없이 확장할 수 있다.
- 다형적 메서드 호출: 동일한 메서드 이름으로 다양한 동작을 정의하여 재사용성을 높인다.

계층 구조를 활용한 다형성

ATM 클래스는 Transaction 포인터를 통해 거래 객체를 다룬다. 이를 통해 입금, 출금, 송금 등의 다양한 거래 유형을 하나의 인터페이스로 처리할 수 있다.

구현 예시

```
void processTransactions(const vector<Transaction*>& transactions) {
    for (const auto& transaction : transactions) {
        transaction->process(); // Polymorphism을 활용하여 동적 바인딩
```

```
        }
    }
```

~\Downloads\ATM_System_OOP_HotPeppersBank-main\ATM_System_OOP_HotPeppersBank-main\ATM.cpp

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

using namespace std;

class ATM;

class User {
protected:
    string username;
    string password;

public:

    User(const string& username, const string& password)
        : username(username), password(password) {
    }

    User() : username(""), password("") {}


    string getUserName() const { return username; }
    string getPassword() const { return password; }


    void setUserName(const string& username) { this->username = username; }
    void setPassword(const string& password) { this->password = password; }
};


class Transaction {
private:
    int transactionID;
    string cardNumber;
    string accountNumber;
    string transactionType;
    int amount;
    string additionalInfo;

public:
    static int transaction_counter;
    Transaction(int id, const string& accountNb, const string& card, const string& type, int
amt, const string& info = "")
```

```
46          : transactionID(id), accountNumber(accountNb), cardNumber(card),
    transactionType(type), amount(amt), additionalInfo(info) {
47      }
48
49
50      virtual ~Transaction() {}
51
52
53      int getTransactionID() const { return transactionID; }
54      string getCardNumber() const { return cardNumber; }
55      string getAccountNumber() const { return accountNumber; }
56      string getTransactionType() const { return transactionType; }
57      int getAmount() const { return amount; }
58      string getAdditionalInfo() const { return additionalInfo; }
59  };
60
61  vector<Transaction*> allTransactions;
62  class DepositTransaction : public Transaction {
63  public:
64      DepositTransaction(int id, const string& accountNb, const string& card, int amt)
65          : Transaction(id, accountNb, card, "Deposit", amt, "Won") {
66      }
67      DepositTransaction(const string& accountNb, const string& card, int amt)
68          : Transaction(0, accountNb, card, "입금", amt, "원") {
69      }
70  };
71
72  class WithdrawTransaction : public Transaction {
73  public:
74      WithdrawTransaction(int id, const string& accountNb, const string& card, int amt)
75          : Transaction(id, accountNb, card, "Withdraw", amt, "Won") {
76      }
77      WithdrawTransaction(const string& accountNb, const string& card, int amt)
78          : Transaction(0, accountNb, card, "출금", amt, "원") {
79      }
80  };
81
82  class TransferTransaction : public Transaction {
83  private:
84      string targetAccountNumber;
85
86  public:
87      TransferTransaction(int id, const string& accountNb, const string& card, int amt, const
    string& target_account)
88          : Transaction(id, accountNb, card, "Transfer", amt, "Won, target account: " +
    target_account) {
89      }
90      TransferTransaction(const string& accountNb, const string& card, int amt, const string&
    target_account)
```

```
91          : Transaction(0, accountNb, card, "계좌이체", amt, "원 도착 계좌: " + target_account)
{
92      }
93
94      string getTargetAccountNumber() const { return targetAccountNumber; }
95  };
96
97
98  class Account : public User {
99  private:
100     string bankName;
101     string cardNumber;
102     string AccountNumber;
103     int balance;
104     vector<Transaction> transaction;
105
106 public:
107     Account(const string& bank_name, const string& username, const string& card_number,
108         const string& account_number, const string& password, int initial_balance)
109         : User(username, password), bankName(bank_name), cardNumber(card_number),
110         AccountNumber(account_number), balance(initial_balance) {
111     }
112
113     Account() : User(), bankName(""), cardNumber(""), AccountNumber(""), balance(0) {}
114
115     string getcardNumber() const { return cardNumber; }
116     string getAccountNumber() const { return AccountNumber; }
117     int getBalance() const { return balance; }
118     string getBankName() const { return bankName; }
119
120     void setBalance(int money) { this->balance = money; }
121 };
122
123 class Bank {
124 private:
125     string name;
126     unordered_map<string, Account> accounts;
127
128 public:
129     Bank(const string& bankName) : name(bankName) {}
130     string getName() const { return name; }
131
132     void addAccount(const Account& account) {
133         accounts[account.getcardNumber()] = account;
134     }
135
136     Account* authenticate(const string& cardNumber, const string& password) {
137         if (accounts.count(cardNumber) && accounts[cardNumber].getPassword() == password) {
138             return &accounts[cardNumber];
```

```
        }

        return nullptr;
    }


    Account* findAccountByCardNumber(const string& cardNumber) {
        if (accounts.count(cardNumber)) {
            return &accounts[cardNumber];
        }
        return nullptr;
    }

    string getCardNumByAccNum(const string& accountNumber) const {
        for (const auto& entry : accounts) {
            const Account& account = entry.second;
            if (account.getAccountNumber() == accountNumber) {
                return account.getcardNumber();
            }
        }
        return "0";
    }

    Account* findAccountByAccountNumber(const string& AccountNumber) {

        if (accounts.count(AccountNumber)) {
            return &accounts[AccountNumber];
        }
        return nullptr;
    }

    void displayAccounts() const {
        for (const auto& entry : accounts) {
            const Account& account = entry.second;
            cout << "Account [Bank: " << name
                << ", Card Number: " << account.getcardNumber()
                << ", Username: " << account.getUserName()
                << "] balance: " << account.getBalance() << "Won\n";
        }
    }
};

// 2024/11/21 고침
Bank* findBankByName(const string& bankName, const vector<Bank*>& allBanks) {
    for (Bank* bank : allBanks) {
        if (bank->getName() == bankName) {
            return bank;
        }
    }
```

```
188     return nullptr;
189 }
190 
191 
192 void displaySnapshot(const vector<Bank>& banks, const vector<ATM>& atms);
193 
194 class ATM {
195 private:
196     string serialNumber;
197     string atmType;
198     string language;
199     Bank* primaryBank;
200     int cash1000, cash5000, cash10000, cash50000;
201     string adminCardNumber = "0000";
202     bool session_active = false;
203     vector<Transaction*> session_transactions;
204 
205 
206 public:
207     ATM(string type, Bank* bank, string serial, string lang, int c1000, int c5000, int
    c10000, int c50000)
208         : atmType(type), primaryBank(bank), serialNumber(serial), language(lang),
209         cash1000(c1000), cash5000(c5000), cash10000(c10000), cash50000(c50000) {
210     }
211     ~ATM() {
212         for (Transaction* transaction : session_transactions) {
213             delete transaction;
214         }
215     }
216     string getSerialNumber() const {
217         return serialNumber;
218     }
219 
220     bool authenticateUser(const vector<Bank*>& allBanks, const string& cardNumber, string&
    password, int language_signal) {
221         int attemptCount = 0;
222         while (attemptCount < 3) {
223             if (atmType == "Single") {
224                 Account* account = primaryBank->authenticate(cardNumber, password);
225                 if (account) {
226                     if (language_signal == 1) {
227                         cout << "Access granted for Single-Bank ATM. You are authorized to
    access " << primaryBank->getName() << " Bank." << endl;
228                     }
229                     else {
230                         cout << "단일 은행 ATM에 대한 접근 권한이 부여되었습니다. " <<
    primaryBank->getName() << " 은행에 접근할 수 있습니다." << endl;
231                     }
232                     return true;
233                 }
```

```
234                 else if (primaryBank->findAccountByCardNumber(cardNumber)) {
235
236                     if (language_signal == 1) {
237                         cout << "Incorrect password. Please try again (" << 3 - attemptCount
    - 1 << " attempts left).\n";
238                     }
239                     else {
240                         cout << "비밀번호가 틀렸습니다. 다시 시도하십시오. (" << 3 -
    attemptCount - 1 << "회 남음)\n";
241                     }
242                 }
243                 else {
244                     if (language_signal == 1) {
245                         cout << "Invalid card. Please note that this ATM is associated with
    " << primaryBank->getName() << " Bank." << endl;
246                     }
247                     else {
248                         cout << "유효하지 않은 카드입니다. 이 ATM은 " << primaryBank-
    >getName() << " 은행과 연결되어 있음을 참고하십시오." << endl;
249                     }
250                     return false;
251                 }
252             }
253             else if (atmType == "Multi") {
254                 for (Bank* bank : allBanks) {
255                     Account* account = bank->authenticate(cardNumber, password);
256                     if (account) {
257                         if (language_signal == 1) {
258                             cout << "Access granted for Multi-Bank ATM. You are authorized
    to access " << bank->getName() << " Bank." << endl;
259                         }
260                         else {
261                             cout << "다중 은행 ATM에 대한 접근 권한이 부여되었습니다. " <<
    bank->getName() << " 은행에 접근할 수 있습니다." << endl;
262                         }
263                         return true;
264                     }
265                 }
266                 if (language_signal == 1) {
267                     cout << "Authorization failed. Please try again (" << 3 - attemptCount -
    1 << " attempts remaining).\n";
268                 }
269                 else {
270                     cout << "인증 실패. 다시 시도하십시오. (" << 3 - attemptCount - 1 << "회
    남음)\n";
271                 }
272             }
273
274             attemptCount++;
275             if (attemptCount < 3) {
```

```
276                 cout << "Please re-enter your password: ";
277                 cin >> password;
278             }
279         }
280         if (language_signal == 1) {
281             cout << "Maximum attempts reached. Session terminated for security.\n";
282         }
283         else {
284             cout << "최대 시도 횟수에 도달했습니다. 보안을 위해 세션이 종료되었습니다.\n";
285         }
286         return false;
287     }
288 
289 
290     void displayRemainingCash() const {
291         cout << "ATM [SerialNum: " << serialNumber << "] remaining cash: {"
292             << "KRW 50000 : " << cash50000 << ", "
293             << "KRW 10000 : " << cash10000 << ", "
294             << "KRW 5000 : " << cash5000 << ", "
295             << "KRW 1000 : " << cash1000 << "}\n";
296     }
297 
298     void deposit(Account* account, Bank* userBank, int language_signal) {
299         int depositType;
300         if (language_signal == 1) {
301             cout << "\n<< Choose deposit type >>\n1. Cash deposit\n2. Check deposit\n3.
    Cancel (press any key)\n-> ";
302         }
303         else {
304             cout << "\n<< 입금 유형 선택 >>\n1. 현금 입금\n2. 수표 입금\n3. 취소 (아무 키나 누르
    세요)\n-> ";
305         }
306         cin >> depositType;
307 
308         if (depositType != 1 && depositType != 2) {
309             if (language_signal == 1) {
310                 cout << "Deposit has been canceled.\n";
311             }
312             else {
313                 cout << "입금이 취소되었습니다.\n";
314             }
315             return;
316         }
317 
318         bool isCashDeposit = (depositType == 1);
319         bool isCheckDeposit = (depositType == 2);
320 
321         bool requiresFee = (primaryBank->getName() != userBank->getName());
322         int fee = requiresFee ? 2000 : 1000;
```

```
323
324
325
326         if (language_signal == 1) {
327             cout << "The deposit fee is KRW " << fee << ".\nWould you like to proceed?\n1.
    Yes\n2. Cancel (press any key)\n-> ";
328         }
329         else {
330             cout << "입금 수수료는 " << fee << "원입니다. 수수료를 지불하시겠습니까?\n1. 예\n2.
    취소 (아무 키나 누르세요)\n-> ";
331         }
332         int confirm;
333         cin >> confirm;
334
335         if (confirm != 1) {
336             if (language_signal == 1) {
337                 cout << "Deposit has been canceled.\n";
338             }
339             else {
340                 cout << "입금이 취소되었습니다.\n";
341             }
342             return;
343         }
344
345
346         if (fee == 1000) {
347             cash1000++;
348             if (language_signal == 1) {
349                 cout << "A fee of KRW 1,000 has been accepted.\n";
350             }
351             else {
352                 cout << "1,000원의 수수료가 처리되었습니다.\n";
353             }
354         }
355         else if (fee == 2000) {
356             cash1000 += 2;
357             if (language_signal == 1) {
358                 cout << "A fee of KRW 2,000 has been accepted.\n";
359             }
360             else {
361                 cout << "2,000원의 수수료가 처리되었습니다.\n";
362             }
363         }
364
365         if (isCashDeposit) {
366
367             int count1000 = 0, count5000 = 0, count10000 = 0, count50000 = 0;
368             int totalBills = 0;
369
```

```
370
371             auto getPositiveInteger = [&](const string& prompt) -> int {
372                 int count;
373                 while (true) {
374                     cout << prompt;
375                     cin >> count;
376
377                     if (cin.fail() || count < 0) {
378                         cin.clear();
379                         cin.ignore(1000, '\n');
380                         if (language_signal == 1) {
381                             cout << "Invalid input. Please enter a non-negative integer.\n";
382                         }
383                         else {
384                             cout << "잘못된 입력입니다. 양의 정수를 입력하세요.\n";
385                         }
386                     }
387                     else {
388                         return count;
389                     }
390                 }
391                 };
392
393
394             if (language_signal == 1) {
395                 count1000 = getPositiveInteger("Enter the number of KRW 1,000 bills you want
    to deposit: ");
396             }
397             else {
398                 count1000 = getPositiveInteger("천 원권 몇 장을 입금하시겠습니까? ");
399             }
400             totalBills += count1000;
401
402
403             if (language_signal == 1) {
404                 count5000 = getPositiveInteger("Enter the number of KRW 5,000 bills you want
    to deposit: ");
405             }
406             else {
407                 count5000 = getPositiveInteger("오천 원권 몇 장을 입금하시겠습니까? ");
408             }
409             totalBills += count5000;
410
411
412             if (language_signal == 1) {
413                 count10000 = getPositiveInteger("Enter the number of KRW 10,000 bills you
    want to deposit: ");
414             }
415             else {
```

```
416                 count10000 = getPositiveInteger("만원권 몇 장을 입금하시겠습니까? ");
417             }
418             totalBills += count10000;
419 
420 
421             if (language_signal == 1) {
422                 count50000 = getPositiveInteger("Enter the number of KRW 50,000 bills you
    want to deposit: ");
423             }
424             else {
425                 count50000 = getPositiveInteger("오만 원권 몇 장을 입금하시겠습니까? ");
426             }
427             totalBills += count50000;
428 
429 
430             if (language_signal == 1) {
431                 cout << "Total number of bills deposited: " << totalBills << "\n";
432             }
433             else {
434                 cout << "총 입금한 지폐 수: " << totalBills << "\n";
435             }
436 
437 
438 
439             const int limit = 50;
440             if (totalBills > limit) {
441                 if (language_signal == 1) {
442                     cout << "You cannot deposit more than " << limit << " bills at a time.
    \n!!Session terminated!!\n";
443                 }
444                 else {
445                     cout << limit << "장을 초과하여 입금할 수 없습니다.\n!!세션 종료!!\n";
446                 }
447                 return;
448             }
449 
450 
451             int depositAmount = count1000 * 1000 + count5000 * 5000 + count10000 * 10000 +
    count50000 * 50000;
452 
453             if (depositAmount == 0) {
454                 if (language_signal == 1) {
455                     cout << "Deposit amount cannot be zero. The fee will be refunded.\n";
456                 }
457                 else {
458                     cout << "입금 금액이 0원입니다. 수수료가 환불됩니다.\n";
459                 }
460                 cash1000 -= (fee / 1000);
461                 if (language_signal == 1) {
```

```
462                     cout << "The fee of KRW " << fee << " has been refunded.\n";
463                 }
464                 else {
465                     cout << fee << "원이 환불되었습니다.\n";
466                 }
467                 return;
468             }
469 
470             if (language_signal == 1) {
471                 cout << "The total deposit amount is KRW " << depositAmount << ".\nWould you like to proceed?\n1. Yes\n2. Cancel (press any key)\n-> ";
472             }
473             else {
474                 cout << "입금 금액은 " << depositAmount << "원입니다.\n계속 진행하시겠습니까?\n1. 예\n2. 취소 (아무 키나 누르세요)\n-> ";
475             }
476             cin >> confirm;
477             if (confirm != 1) {
478                 if (language_signal == 1) {
479                     cout << "Deposit has been canceled.\n";
480                 }
481                 else {
482                     cout << "입금이 취소되었습니다.\n";
483                 }
484 
485                 return;
486             }
487 
488 
489             account->setBalance(account->getBalance() + depositAmount);
490             if (language_signal == 1) {
491                 cout << "Deposit successful! Your new balance is KRW " << account->getBalance() << ".\n";
492                 addTransaction(-1, account->getAccountNumber(), account->getcardNumber(), "Deposit", depositAmount, "");
493             }
494             else {
495                 cout << "입금이 완료되었습니다!\n계좌 잔액: " << account->getBalance() << "원\n";
496                 addTransaction(0, account->getAccountNumber(), account->getcardNumber(), "Deposit", depositAmount, "");
497             }
498 
499 
500             cash1000 += count1000;
501             cash5000 += count5000;
502             cash10000 += count10000;
503             cash50000 += count50000;
504 
```

```
505             addTransaction(Transaction::transaction_counter++, account->getAccountNumber(),
    account->getcardNumber(), "Deposit", depositAmount, ""); //11.18 21:36
506         }
507 
508         else if (isCheckDeposit) {
509             int paperNum = 0;
510             double CheckValue = 0;
511             int depositAmount = 0;
512 
513 
514             if (language_signal == 1) {
515                 cout << "Enter the number of checks you want to deposit (Maximum 30): ";
516             }
517             else {
518                 cout << "몇 장의 수표를 입금하시겠습니까? (최대 30장)" << endl;
519             }
520 
521             cin >> paperNum;
522 
523             if (paperNum > 30) {
524                 if (language_signal == 1) {
525                     cout << "You cannot deposit more than 30 checks at a time.\n";
526                 }
527                 else {
528                     cout << "30장을 초과하여 입금할 수 없습니다." << endl;
529                 }
530                 return;
531             }
532             int finalCheck;
533 
534 
535             for (int i = 0; i < paperNum; i++) {
536                 while (true) {
537                     if (language_signal == 1) {
538                         cout << "Check #" << (i + 1) << ": Enter the check amount (Minimum:
    KRW 100,000): ";
539                     }
540                     else {
541                         cout << "수표 #" << (i + 1) << ": 수표 금액을 입력하세요 (최소 100,000
    원 이상)" << endl;
542                     }
543 
544                     cin >> CheckValue;
545 
546                     if (cin.fail() || CheckValue < 0) {
547                         cin.clear();
548 
549                         if (language_signal == 1) {
550                             cout << "Invalid input. Please enter a positive integer amount
    of at least KRW 100,000.\n";
```

```
551                         }
552                         else {
553                             cout << "잘못된 입력입니다. 양의 정수로 100,000원 이상의 금액을 입력
    하세요." << endl;
554                         }
555                     }
556                     else if (CheckValue >= 100000) {
557                         if (language_signal == 1) {
558                             cout << "The Check amount is KRW " << CheckValue << ".\nWould
    you like to proceed?\n1. Yes\n2. Cancel (press any key)\n->";
559                             cin >> finalCheck;
560                             if (finalCheck != 1) {
561                                 return;
562                             }
563 
564                         }
565                         else {
566                             cout << "수표 금액은 " << CheckValue << "원입니다.\n계속 진행하시
    겠습니까?\n1. 예\n2. 취소 (아무 키나 누르세요)\n-> ";
567                             cin >> finalCheck;
568                             if (finalCheck != 1) {
569                                 return;
570                             }
571                         }
572                         depositAmount += CheckValue;
573                         break;
574                     }
575                     else {
576                         if (language_signal == 1) {
577                             cout << "Invalid amount. Please enter an amount of at least KRW
    100,000.\n";
578                         }
579                         else {
580                             cout << "잘못된 금액입니다. 100,000원 이상의 금액을 입력하세요." <<
    endl;
581                         }
582                     }
583                 }
584             }
585 
586 
587             account->setBalance(account->getBalance() + depositAmount);
588             if (language_signal == 1) {
589                 cout << "Deposit successful! Your new balance is KRW " << account-
    >getBalance() << ".\n";
590                 addTransaction(-1, account->getAccountNumber(), account->getcardNumber(),
    "Deposit", depositAmount, "");
591             }
592             else {
```

```
593                 cout << "입금이 완료되었습니다!\n계좌 잔액: " << account->getBalance() << "원
    \n";
594                 addTransaction(0, account->getAccountNumber(), account->getcardNumber(),
    "Deposit", depositAmount, "");
595             }
596
597
598             addTransaction(Transaction::transaction_counter++, account->getAccountNumber(),
    account->getcardNumber(), "Deposit", depositAmount, "");
599         }
600
601     }
602
603
604
605     void withdraw(Account* account, Bank* userBank, int language_signal) {
606         const int maxWithdrawAmount = 500000;
607         const int maxWithdrawalsPerSession = 3;
608         int withdrawalsThisSession = 0;
609
610         while (withdrawalsThisSession < maxWithdrawalsPerSession) {
611             if (language_signal == 1) {
612                 cout << "\nWould you like to proceed with a withdrawal?\n1. Yes\n2. No (End
    withdrawal session)\n-> ";
613             }
614             else {
615                 cout << "\n출금 계속 진행하시겠습니까? \n1. 네\n2. 아니 (출금 세션을 종료)\n->
    ";
616             }
617
618             int proceed;
619             cin >> proceed;
620
621             if (proceed != 1) {
622                 if (language_signal == 1) {
623                     cout << "Withdrawal session has ended.\n";
624                 }
625                 else {
626                     cout << "출금 세션을 종료합니다.\n";
627                 }
628                 return;
629             }
630
631             int withdrawalAmount = 0;
632             if (language_signal == 1) {
633                 cout << "Enter the amount to withdraw: ";
634             }
635             else {
636                 cout << "출금할 금액을 입력하시오: ";
637             }
```

```
638             cin >> withdrawalAmount;
639 
640 
641             if (withdrawalAmount > maxWithdrawAmount) {
642                 if (language_signal == 1) {
643                     cout << "The maximum withdrawal amount per transaction is KRW " <<
    maxWithdrawAmount << ".\n";
644                 }
645                 else {
646                     cout << "한 번에 출금할 수 있는 최대 금액은 KRW " << maxWithdrawAmount <<
    "원입니다.\n";
647                 }
648                 return;
649             }
650 
651 
652             bool requiresFee = (primaryBank->getName() != userBank->getName());
653             int fee = requiresFee ? 2000 : 1000;
654 
655 
656             if (withdrawalAmount + fee > account->getBalance()) {
657                 if (language_signal == 1) {
658                     cout << "Your account balance is insufficient to complete this
    withdrawal.\n";
659                 }
660                 else {
661                     cout << "계좌 잔액이 부족하여 출금할 수 없습니다. \n";
662                 }
663                 return;
664             }
665 
666 
667             int remainingAmount = withdrawalAmount;
668             int withdraw50000 = 0, withdraw10000 = 0, withdraw5000 = 0, withdraw1000 = 0;
669 
670             if (remainingAmount >= 50000 && cash50000 > 0) {
671                 withdraw50000 = std::min(remainingAmount / 50000, cash50000);
672                 remainingAmount -= withdraw50000 * 50000;
673             }
674             if (remainingAmount >= 10000 && cash10000 > 0) {
675                 withdraw10000 = std::min(remainingAmount / 10000, cash10000);
676                 remainingAmount -= withdraw10000 * 10000;
677             }
678             if (remainingAmount >= 5000 && cash5000 > 0) {
679                 withdraw5000 = std::min(remainingAmount / 5000, cash5000);
680                 remainingAmount -= withdraw5000 * 5000;
681             }
682             if (remainingAmount >= 1000 && cash1000 > 0) {
683                 withdraw1000 = std::min(remainingAmount / 1000, cash1000);
```

```
684                 remainingAmount -= withdraw1000 * 1000;
685             }
686 
687 
688             if (remainingAmount > 0) {
689                 if (language_signal == 1) {
690                     cout << "The ATM does not have enough cash to fulfill your withdrawal.
    \n";
691                 }
692                 else {
693                     cout << "ATM에 출금할 현금이 부족합니다.\n";
694                 }
695                 return;
696             }
697 
698 
699             if (language_signal == 1) {
700                 cout << "The withdrawal amount is KRW " << withdrawalAmount
701                     << " with a fee of KRW " << fee << ".\n";
702                 cout << "50000: " << withdraw50000 << ", 10000: " << withdraw10000
703                     << ", 5000: " << withdraw5000 << ", 1000: " << withdraw1000 << "\n";
704                 cout << "Would you like to confirm the withdrawal?\n1. Confirm\n2. Cancel\n-
    > ";
705             }
706             else {
707                 cout << "출금 금액은 " << withdrawalAmount << "원이며, 수수료는 "
708                     << fee << "원입니다.\n";
709                 cout << "50000원권: " << withdraw50000 << "장, 10000원권: " << withdraw10000
710                     << "장, 5000원권: " << withdraw5000 << "장, 1000원권: " << withdraw1000
    << "장\n";
711                 cout << "출금을 확인하시겠습니까?\n1. 확인\n2. 취소\n-> ";
712             }
713 
714             int confirm;
715             cin >> confirm;
716             if (confirm != 1) {
717                 if (language_signal == 1) {
718                     cout << "Withdrawal canceled. Returning to session.\n";
719                 }
720                 else {
721                     cout << "출금이 취소되었습니다. 세션으로 돌아갑니다.\n";
722                 }
723                 return;
724             }
725 
726 
727             account->setBalance(account->getBalance() - (withdrawalAmount + fee));
728             cash50000 -= withdraw50000;
729             cash10000 -= withdraw10000;
```

```
730             cash5000 -= withdraw5000;
731             cash1000 -= withdraw1000;
732 
733 
734             if (language_signal == 1) {
735                 cout << "Successfully withdrawn! Remaining balance: KRW "
736                     << account->getBalance() << "\n";
737                 addTransaction(Transaction::transaction_counter++, account->getAccountNumber(), account->getcardNumber(), "Withdraw", withdrawalAmount, ""); //11.18 21:49
738                 addTransaction(-1, account->getAccountNumber(), account->getcardNumber(), "Withdraw", withdrawalAmount, "");
739 
740             }
741             else {
742                 cout << "출금 성공! 계좌 잔액: KRW " << account->getBalance() << "\n";
743                 addTransaction(Transaction::transaction_counter++, account->getAccountNumber(), account->getcardNumber(), "Withdraw", withdrawalAmount, ""); //11.18 21:49
744                 addTransaction(0, account->getAccountNumber(), account->getcardNumber(), "Withdraw", withdrawalAmount, "");
745             }
746             if (cash1000 + cash5000 + cash10000 + cash50000 == 0) {
747                 return;
748             }
749 
750 
751             withdrawalsThisSession++;
752             if (withdrawalsThisSession >= maxWithdrawalsPerSession) {
753                 if (language_signal == 1) {
754                     cout << "The maximum number of withdrawals per session is "
755                         << maxWithdrawalsPerSession << ". Please restart another session for more withdrawals.\n";
756                 }
757                 else {
758                     cout << "세션당 출금 최대 횟수: " << maxWithdrawalsPerSession
759                         << ". 추가 출금을 하려면 세션을 새로 시작하십시오.\n";
760                 }
761                 break;
762             }
763         }
764     }
765 
766 
767 
768 
769 
770     void transfer(Account* sourceAccount, Bank* sourceBank, const vector<Bank*>& allBanks, int language_signal) {
771         int transferType;
```

```
772         if (language_signal == 1) {
773             cout << "\n<< Choose transfer type >>\n1. Cash transfer\n2. Account transfer\n3.
    Cancel (press any key)\n-> ";
774         }
775         else {
776             cout << "\n<< 송금 유형 선택 >>\n1. 현금 송금\n2. 계좌 송금\n3. 취소 (아무 키나 입
    력)\n-> ";
777         }
778         cin >> transferType;
779 
780         if (transferType == 3) {
781             if (language_signal == 1) {
782                 cout << "Transfer has been canceled.\n";
783             }
784             else {
785                 cout << "송금이 취소되었습니다.\n";
786             }
787             return;
788         }
789 
790 
791         string destinationAccountNumber, destinationCardNumber, destinationBankName;
792         if (language_signal == 1) {
793             cout << "Enter the name of the destination bank: ";
794         }
795         else {
796             cout << "송금 받을 은행 이름을 입력하세요: ";
797         }
798         cin >> destinationBankName;
799 
800 
801         Bank* destinationBank = nullptr;
802         for (Bank* bank : allBanks) {
803             if (bank->getName() == destinationBankName) {
804                 destinationBank = bank;
805                 break;
806             }
807         }
808 
809         if (!destinationBank) {
810             if (language_signal == 1) {
811                 cout << "The destination bank could not be found. Transfer canceled.\n";
812             }
813             else {
814                 cout << "송금 받을 은행을 찾을 수 없습니다. 송금을 취소합니다.\n";
815             }
816             return;
817         }
818 
```

```
819
820         bool isAcc{ false };
821         while (!isAcc) {
822             do {
823                 if (language_signal == 1) {
824                     cout << "Input destination account number (if you want to cancel, press
    0) :";
825                     cin >> destinationAccountNumber;
826                     if (destinationAccountNumber == "0") {
827                         return;
828                     }
829                 }
830                 else {
831                     cout << "도착 계좌번호를 입력하세요 (취소를 원하면 0을 치세요.) :";
832                     cin >> destinationAccountNumber;
833                     if (destinationAccountNumber == "0") {
834                         return;
835                     }
836                 }
837                 if (destinationAccountNumber == sourceAccount->getAccountNumber()) {
838                     if (language_signal == 1) {
839                         cout << "source account and destionation account cannot be the same.
    \n";
840                     }
841                     else {
842                         cout << "송금계좌와 도착계좌가 같은 수는 없습니다.\n";
843                     }
844                 }
845             } while (destinationAccountNumber == sourceAccount->getAccountNumber());
846
847             if (destinationBank->getCardNumByAccNum(destinationAccountNumber) == "0") {
848                 if (language_signal == 1) {
849                     cout << "There is no such account number in the Bank." << endl;
850                 }
851                 else {
852                     cout << "해당 은행에 없는 계좌입니다." << endl;
853                 }
854             }
855             else {
856                 isAcc = true;
857             }
858         };
859
860
861         destinationCardNumber = destinationBank->getCardNumByAccNum(destinationAccountNu↩
    mber);
862
863
864         int transferFee = 0;
```

```
865
866
867         if (sourceBank == primaryBank && destinationBank == primaryBank) {
868             transferFee = 2000;
869         }
870         else if (sourceBank == primaryBank || destinationBank == primaryBank) {
871             transferFee = 3000;
872         }
873         else {
874             transferFee = 4000;
875         }
876
877         int transferAmount = 0;
878
879         if (transferType == 1) {
880
881             int cash1000, cash5000, cash10000, cash50000;
882             int command;
883             int insertedAmount;
884             transferFee = 1000;
885
886             if (language_signal == 1) {
887                 cout << "Cash transfer fee costs KRW " << transferFee << ". Please insert
    the same amount of cash to the ATM.\n";
888                 cout << "1. Confirm\n2. Cancel\n-> ";
889             }
890             else {
891                 cout << "송금 수수료는 KRW " << transferFee << " 원입니다. 수수료만큼의 현금을
    투입해주세요. \n";
892                 cout << "1. 확인\n2. 취소\n-> ";
893             }
894             cin >> command;
895
896             if (command == 2) {
897                 if (language_signal == 1) {
898                     cout << "Transaction has been canceled.\n";
899                 }
900                 else {
901                     cout << "송금이 취소되었습니다.\n";
902                 }
903                 return;
904             }
905
906             if (language_signal == 1) {
907                 cout << "Please insert the cash to complete the transfer.\n";
908             }
909             else {
910                 cout << "송금을 완료하려면 송금할 금액을 입력하십시오.\n";
911             }
```

```
912
913             do {
914                 if (language_signal == 1) {
915                     cout << "Enter the number of KRW 1,000 bills: ";
916                 }
917                 else {
918                     cout << "천 원권 개수: ";
919                 }
920                 cin >> cash1000;
921
922                 if (language_signal == 1) {
923                     cout << "Enter the number of KRW 5,000 bills: ";
924                 }
925                 else {
926                     cout << "오천 원권 개수: ";
927                 }
928                 cin >> cash5000;
929
930                 if (language_signal == 1) {
931                     cout << "Enter the number of KRW 10,000 bills: ";
932                 }
933                 else {
934                     cout << "만원권 개수: ";
935                 }
936                 cin >> cash10000;
937
938                 if (language_signal == 1) {
939                     cout << "Enter the number of KRW 50,000 bills: ";
940                 }
941                 else {
942                     cout << "오만 원권 개수: ";
943                 }
944                 cin >> cash50000;
945
946                 insertedAmount = cash1000 * 1000 + cash5000 * 5000 + cash10000 * 10000 +
    cash50000 * 50000;
947
948                 if (language_signal == 1) {
949                     cout << "The total amount entered is: " << insertedAmount << " KRW.\n";
950                     cout << "1. Confirm\n2. Re-enter cash\n-> ";
951                 }
952                 else {
953                     cout << "총 입력 금액: " << insertedAmount << "원.\n";
954                     cout << "1. 확인\n2. 다시 입력\n-> ";
955                 }
956                 cin >> command;
957
958
959             } while (command != 1);
```

```
960
961
962             this->cash1000 += (cash1000 + 1);
963             this->cash5000 += cash5000;
964             this->cash10000 += cash10000;
965             this->cash50000 += cash50000;
966
967
968             transferAmount = insertedAmount;
969
970
971             if (language_signal == 1) {
972                 cout << "Successfully transferred " << transferAmount << " KRW.\n";
973                 addTransaction(-1, sourceAccount->getAccountNumber(), sourceAccount-
    >getcardNumber(), "Transfer", transferAmount, destinationAccountNumber);
974             }
975             else {
976                 cout << transferAmount << " 원이 성공적으로 송금되었습니다. \n";
977                 addTransaction(0, sourceAccount->getAccountNumber(), sourceAccount-
    >getcardNumber(), "Transfer", transferAmount, destinationAccountNumber);
978             }
979             addTransaction(Transaction::transaction_counter++, sourceAccount-
    >getAccountNumber(), sourceAccount->getcardNumber(), "Transfer", transferAmount,
    destinationAccountNumber);
980         }
981
982         else if (transferType == 2) {
983
984
985             if (language_signal == 1) {
986                 cout << "Enter the amount you wish to transfer: ";
987             }
988             else {
989                 cout << "송금할 금액을 입력하세요: ";
990             }
991             cin >> transferAmount;
992
993
994             if (sourceAccount->getBalance() < transferAmount + transferFee) {
995                 if (language_signal == 1) {
996                     cout << "Insufficient funds in your account. Transfer canceled.\n";
997                 }
998                 else {
999                     cout << "계좌에 잔액이 부족합니다. 송금이 취소됩니다.\n";
1000                }
1001                return;
1002            }
1003
1004
1005            if (language_signal == 1) {
```

```
1006                cout << "Your transfer fee is KRW " << transferFee << ". \nWould you like to
    proceed?\n1. Yes\n2. Cancel (press any key)\n-> ";
1007                int confirm;
1008                cin >> confirm;
1009                if (confirm != 1) {
1010                    cout << "Transfer canceled.\n";
1011                    return;
1012                }
1013            }
1014            else {
1015                cout << "수수료 : KRW " << transferFee << ".송금 진행하시겠습니까?\n1. 네\n2.
    아니요 (아무키나 입력)\n-> ";
1016                int confirm;
1017                cin >> confirm;
1018                if (confirm != 1) {
1019                    cout << "송금 취소.\n";
1020                    return;
1021                }
1022            }
1023            sourceAccount->setBalance(sourceAccount->getBalance() - transferAmount -
    transferFee);
1024        }
1025
1026
1027        Account* destinationAccount = destinationBank->findAccountByCardNum↩
    ber(destinationCardNumber);
1028        if (destinationAccount) {
1029            destinationAccount->setBalance(destinationAccount->getBalance() +
    transferAmount);
1030            if (language_signal == 1) {
1031                cout << "Successfully transferred " << transferAmount << " KRW to account "
    << destinationAccountNumber << ".\n";
1032            }
1033            else {
1034                cout << destinationAccountNumber << " 계좌에 " << transferAmount << "원이 성
    공적으로 송금되었습니다.\n";
1035            }
1036            if (transferType == 2) {
1037                if (language_signal == 1) {
1038                    cout << "There is KRW " << sourceAccount->getBalance() << " in your
    account.\n";
1039                    addTransaction(Transaction::transaction_counter++, sourceAccount-
    >getAccountNumber(), sourceAccount->getcardNumber(), "Transfer", transferAmount,
    destinationAccountNumber); //11.18 21:52
1040                    addTransaction(-1, sourceAccount->getAccountNumber(), sourceAccount-
    >getcardNumber(), "Transfer", transferAmount, destinationAccountNumber);
1041                }
1042                else {
1043                    cout << "당신 계좌에" << sourceAccount->getBalance() << "원이 남아있습니
    다. \n";
```

```
1044                    addTransaction(Transaction::transaction_counter++, sourceAccount->getAccountNumber(), sourceAccount->getcardNumber(), "Transfer", transferAmount, destinationAccountNumber); //11.18 21:52
1045                    addTransaction(0, sourceAccount->getAccountNumber(), sourceAccount->getcardNumber(), "Transfer", transferAmount, destinationAccountNumber);
1046                }
1047            }
1048        }
1049        else {
1050            if (language_signal == 1) {
1051                cout << "Destination account could not be found. Transfer canceled.\n";
1052            }
1053            else {
1054                cout << "도착 계좌가 조회되지 않습니다. 송금이 취소됩니다.\n";
1055            }
1056        }
1057    }
1058
1059
1060
1061
1062    void startSession(const vector<Bank*>& allBanks, const vector<Bank>& banks, const vector<ATM>& atms) {
1063        if (cash1000 == 0 && cash5000 == 0 && cash10000 == 0 && cash50000 == 0) {
1064            cout << "ATM has no cash available. Session terminated.\n";
1065            return;
1066        } //WH 11.22 23:26
1067
1068        string cardNumber, password;
1069        int language_signal = 1;
1070
1071
1072        if (language == "Bi") {
1073            cout << "\nSelect language / 언어를 선택하세요:\n1. English\n2. 한국어\n-> ";
1074            cin >> language_signal;
1075        }
1076
1077        while (true) {
1078            if (language_signal == 1) {
1079                cout << "\nPlease Enter your Card number: ";
1080            }
1081            else {
1082                cout << "\n카드 번호를 입력해주세요: ";
1083            }
1084            cin >> cardNumber;
1085
1086            if (cardNumber == adminCardNumber) {
1087                if (language_signal == 1) {
1088                    cout << "Admin session has been started.\n";
1089                }
```

```
1090                 else {
1091                     cout << "관리자 세션이 시작되었습니다.\n";
1092                 }
1093                 return startAdminSession(allTransactions);
1094             }
1095 
1096 
1097             bool accountExists = false;
1098             for (Bank* bank : allBanks) {
1099                 if (bank->findAccountByCardNumber(cardNumber)) {
1100                     accountExists = true;
1101                     break;
1102                 }
1103             }
1104 
1105             if (!accountExists) {
1106                 if (language_signal == 1) {
1107                     cout << "There's no account available with that number. Please try
     again.\n";
1108                 }
1109                 else {
1110                     cout << "해당 번호의 계좌가 존재하지 않습니다. 다시 시도하세요.\n";
1111                 }
1112                 continue;
1113             }
1114             break;
1115         }
1116 
1117         if (language_signal == 1) {
1118             cout << "Enter Password for your account : ";
1119         }
1120         else {
1121             cout << "비밀번호를 입력하세요: ";
1122         }
1123 
1124         cin >> password;
1125 
1126         if (authenticateUser(allBanks, cardNumber, password, language_signal)) {
1127             Account* authenticatedAccount = nullptr;
1128             Bank* cardBank = nullptr;
1129 
1130 
1131             for (Bank* bank : allBanks) {
1132                 authenticatedAccount = bank->authenticate(cardNumber, password);
1133                 if (authenticatedAccount) {
1134                     cardBank = bank;
1135                     if (language_signal == 1) {
1136                         cout << "Authentication successful. Welcome, " <<
     authenticatedAccount->getUserName() << "!\n";
```

```
1137                    }
1138                    else {
1139                        cout << "인증 성공. 환영합니다, " << authenticatedAccount->getUserName() << "!\n";
1140                    }
1141                    break;
1142                }
1143            }
1144
1145            if (!authenticatedAccount) {
1146                if (language_signal == 1) {
1147                    cout << "Authentication has failed. Ending session.\n";
1148                }
1149                else {
1150                    cout << "인증에 실패했습니다. 세션을 종료합니다.\n";
1151                }
1152                return;
1153            }
1154
1155            session_transactions.clear();
1156            while (true) {
1157
1158                if (cash1000 + cash5000 + cash10000 + cash50000 == 0) {
1159                    if (language_signal == 1) {
1160                        cout << "ATM has run out of cash. Ending session.\n";
1161                    }
1162                    else {
1163                        cout << "ATM에 현금이 모두 소진되었습니다. 세션을 종료합니다.\n";
1164                    }
1165                    return;
1166                }
1167                char choice;
1168                bool validInput = false;
1169
1170                while (!validInput) {
1171                    if (language_signal == 1) {
1172                        cout << "\nSelect Transaction:\n1. Deposit\n2. Withdraw\n3. Transfer\n4. Exit\n:";
1173                    }
1174                    else {
1175                        cout << "\n거래를 선택하세요:\n1. 입금\n2. 출금\n3. 송금\n4. 종료\n:";
1176                    }
1177
1178                    cin >> choice;
1179
1180                    if (choice == '1' || choice == '2' || choice == '3' || choice == '4' || choice == '/') {
1181                        validInput = true;
1182                    }
```

```
1183                    else {
1184                        cin.clear();
1185                        cin.ignore(1000, '\n');
1186                        if (language_signal == 1) {
1187                            cout << "Invalid choice. Please enter a valid option (1-4 or
    '/').\n";
1188                        }
1189                        else {
1190                            cout << "잘못된 선택입니다. 유효한 옵션(1-4 또는 '/')을 입력하세요.
    \n";
1191                        }
1192                    }
1193                }
1194
1195                if (choice == '4') break;
1196
1197
1198                switch (choice) {
1199                case '1':
1200                    deposit(authenticatedAccount, cardBank, language_signal);
1201                    break;
1202                case '2':
1203                    withdraw(authenticatedAccount, cardBank, language_signal);
1204                    break;
1205                case '3':
1206                    transfer(authenticatedAccount, cardBank, allBanks, language_signal);
1207                    break;
1208                case '/':
1209                    displaySnapshot(banks, atms);
1210                }
1211            }
1212
1213
1214            if (!session_transactions.empty()) {
1215                if (language_signal == 1) {
1216                    cout << "\nTransaction Summary:\n";
1217                    for (const auto& transaction : session_transactions) {
1218                        cout << "Card Number: " << transaction->getCardNumber()
1219                            << ", Account Number: " << transaction->getAccountNumber()
1220                            << ", Type: " << transaction->getTransactionType()
1221                            << ", Amount: " << transaction->getAmount();
1222                        if (!transaction->getAdditionalInfo().empty()) {
1223                            cout << transaction->getAdditionalInfo();
1224                        }
1225                        cout << endl;
1226                    }
1227                }
1228                else {
1229                    cout << "\n거래 요약:\n";
```

```
1230                    for (const auto& transaction : session_transactions) {
1231                        cout << "카드 번호: " << transaction->getCardNumber()
1232                            << ", 계좌 번호: " << transaction->getAccountNumber()
1233                            << ", 거래 타입: " << transaction->getTransactionType()
1234                            << ", 거래량: " << transaction->getAmount();
1235                        if (!transaction->getAdditionalInfo().empty()) {
1236                            cout << transaction->getAdditionalInfo();
1237                        }
1238                        cout << endl;
1239                    }
1240                }
1241            }
1242            else {
1243                if (language_signal == 1) {
1244                    cout << "\nNo transactions completed in this session.\n";
1245                }
1246                else {
1247                    cout << "\n이번 세션에서 완료된 거래가 없습니다.\n";
1248                }
1249            }
1250        }
1251    }
1252
1253
1254    void startAdminSession(vector<Transaction*>& allTransactions) {
1255        string command;
1256        cout << "Admin session started. Displaying Transaction History.\n";
1257        cout << "Would you like to view the Transaction History? (Enter 'yes') :";
1258        cin >> command;
1259        if (command == "yes") {
1260            displayTransactionHistory(allTransactions);
1261        }
1262        else {
1263            return;
1264        }
1265    }
1266
1267
1268    void addTransaction(int id, const string& account, const string& card, const string&
    type, int amt, const string& info = "", vector<Transaction*>& allTransactions =
    ::allTransactions) {
1269        if (id == 0) {
1270            if (type == "Deposit") {
1271                session_transactions.push_back(new DepositTransaction(account, card, amt));
1272            }
1273            else if (type == "Withdraw") {
1274                session_transactions.push_back(new WithdrawTransaction(account, card, amt));
1275            }
1276            else if (type == "Transfer") {
```

```
1277                session_transactions.push_back(new TransferTransaction(account, card, amt,
     info));
1278            }
1279        }
1280        else if (id == -1) {
1281            if (type == "Deposit") {
1282                session_transactions.push_back(new DepositTransaction(0, account, card,
     amt));
1283            }
1284            else if (type == "Withdraw") {
1285                session_transactions.push_back(new WithdrawTransaction(0, account, card,
     amt));
1286            }
1287            else if (type == "Transfer") {
1288                session_transactions.push_back(new TransferTransaction(0, account, card,
     amt, info));
1289            }
1290
1291        }
1292        else {
1293            if (type == "Deposit") {
1294                allTransactions.push_back(new DepositTransaction(id, account, card, amt));
1295            }
1296            else if (type == "Withdraw") {
1297                allTransactions.push_back(new WithdrawTransaction(id, account, card, amt));
1298            }
1299            else if (type == "Transfer") {
1300                allTransactions.push_back(new TransferTransaction(id, account, card, amt,
     info));
1301            }
1302        }
1303
1304    }
1305    void displayTransactionHistory(const vector<Transaction*>& allTransactions) {
1306        cout << "\n===== Transaction History =====\n";
1307
1308
1309        for (const auto& transaction : allTransactions) {
1310            cout << "ID: " << transaction->getTransactionID()
1311                << ", Account Number: " << transaction->getAccountNumber()
1312                << ", Card: " << transaction->getCardNumber()
1313                << ", Type: " << transaction->getTransactionType()
1314                << ", Amount: " << transaction->getAmount();
1315            if (!transaction->getAdditionalInfo().empty()) {
1316                cout << transaction->getAdditionalInfo();
1317            }
1318            cout << endl;
1319        }
1320
1321
```

```
1322        ofstream outFile("transaction_history.txt");
1323        for (const auto& transaction : allTransactions) {
1324            outFile << "ID: " << transaction->getTransactionID()
1325                << ", Account Number: " << transaction->getAccountNumber()
1326                << ", Card: " << transaction->getCardNumber()
1327                << ", Type: " << transaction->getTransactionType()
1328                << ", Amount: " << transaction->getAmount();
1329            if (!transaction->getAdditionalInfo().empty()) {
1330                outFile << transaction->getAdditionalInfo();
1331            }
1332            outFile << endl;
1333        }
1334        outFile.close();
1335
1336        cout << "Transaction history has been saved to transaction_history.txt text file.
    \n";
1337    }
1338
1339 };
1340
1341 void displaySnapshot(const vector<Bank>& banks, const vector<ATM>& atms) {
1342     cout << "\n\n======< 'Account/ATM Snapshot' >======\n";
1343
1344
1345     cout << "\n[ATM Information - Remaining Cash]\n";
1346     for (const auto& atm : atms) {
1347         atm.displayRemainingCash();
1348     }
1349
1350
1351     cout << "\n[Account Information - Remaining Balance]\n";
1352     for (const auto& bank : banks) {
1353         bank.displayAccounts();
1354     }
1355     cout << "=====================================\n";
1356 }
1357
1358 int Transaction::transaction_counter = 1;
1359
1360
1361 int main() {
1362
1363     cout << "======< 'Bank Initialization' >======\n";
1364     vector<Bank> banks;
1365     vector<Bank*> allBanks;
1366     string bankInput;
1367     vector<string> banknames;
1368     bool vaildbankname;
1369     banks.reserve(100);
```

```
1370     vector<Transaction*> allTransactions;
1371
1372
1373
1374     while (true) {
1375
1376         cout << "Enter bank name for initialization(or type 'done' to finish): ";
1377         cin >> bankInput;
1378         vaildbankname = true;
1379
1380         for (const string& bankname : banknames) {
1381             if (bankInput == bankname) {
1382                 cout << "This bank name has already been taken. Please try to enter a
    different name.\n";
1383                 vaildbankname = false;
1384                 break;
1385             }
1386         }
1387         if (vaildbankname == false) {
1388             continue;
1389         }
1390         banknames.push_back(bankInput);
1391
1392
1393         if (bankInput == "done") {
1394             if (banks.empty()) {
1395                 cout << "No banks has been created. Exiting program.\n";
1396                 return 0;
1397             }
1398             else {
1399                 cout << "Bank initialization has been completed. Proceeding to account
    Initializaiton...\n";
1400                 break;
1401             }
1402         }
1403
1404         Bank newBank(bankInput);
1405         banks.push_back(newBank);
1406         allBanks.push_back(&banks.back());
1407         cout << "[" << bankInput << "] bank has been created!\n";
1408     }
1409
1410
1411
1412     cout << "\n======< 'Account Creation' >======\n";
1413     string createAccountInput;
1414     int accountCount = 1;
1415     vector<string> cardNumbers;
1416     vector<string> accountNumbers;
```

```
1417      cardNumbers.push_back("0000");
1418
1419      do {
1420          cout << "\nWould you like to create Account " << accountCount++ << "? (yes or no) ->
      ";
1421          cin >> createAccountInput;
1422
1423          while (createAccountInput != "yes" && createAccountInput != "no") {
1424              cout << "Invalid input. Please type 'yes' or 'no': ";
1425              cin >> createAccountInput;
1426          }
1427
1428          if (createAccountInput == "yes") {
1429              string bankName, username, cardNumber, AccountNumber, password;
1430              int balance;
1431
1432              bool bankFound = false;
1433              cout << "Bank Name: ";
1434              while (!bankFound) {
1435                  cin >> bankName;
1436                  for (Bank* bank : allBanks) {
1437                      if (bank->getName() == bankName) {
1438                          bankFound = true;
1439                          break;
1440                      }
1441                  }
1442                  if (!bankFound) {
1443                      cout << "Bank not found. Please type again: ";
1444                  }
1445              }
1446
1447              cout << "User Name: ";
1448              cin >> username;
1449
1450
1451              while (true) {
1452                  cout << "Card number (4 digits): ";
1453                  cin >> cardNumber;
1454
1455                  bool validcardNumber = true;
1456
1457                  if (cardNumber.length() != 4) {
1458                      cout << "Invalid Card number. It must be exactly 4 digits.\n";
1459                      validcardNumber = false;
1460                  }
1461
1462
1463                  for (const string& name : cardNumbers) {
1464                      if (cardNumber == name) {
```

```
1465                        validcardNumber = false;
1466                        cout << "This Card number has already been taken. Please try to
    enter a different number.\n";
1467                        break;
1468                    }
1469                }
1470
1471                if (validcardNumber) {
1472                    cardNumbers.push_back(cardNumber);
1473
1474                    break;
1475                }
1476            }
1477
1478
1479            while (true) {
1480                cout << "Account Number (12 digits): ";
1481                cin >> AccountNumber;
1482
1483                bool validAccountNumber = true;
1484
1485
1486                if (AccountNumber.length() != 12) {
1487                    cout << "Invalid Account number. It must be exactly 12 digits.\n";
1488                    validAccountNumber = false;
1489                }
1490
1491
1492                for (const string& existingAccNumber : accountNumbers) {
1493                    if (AccountNumber == existingAccNumber) {
1494                        validAccountNumber = false;
1495                        cout << "This Account number has already been taken. Please try to
    enter a different number.\n";
1496                        break;
1497                    }
1498                }
1499
1500                if (validAccountNumber) {
1501                    accountNumbers.push_back(AccountNumber);
1502
1503                    break;
1504                }
1505            }
1506
1507
1508            while (true) {
1509                try {
1510                    cout << "Balance(KRW): ";
1511                    cin >> balance;
```

```
1512
1513                    if (cin.fail()) {
1514                        throw runtime_error("Invalid input. Please enter a positive
    integer.");
1515                    }
1516
1517                    if (balance < 0) {
1518                        throw runtime_error("Balance cannot be negative.");
1519                    }
1520                }
1521                catch (const runtime_error& e) {
1522                    cin.clear();
1523                    cin.ignore(1000, '\n');
1524                    cout << "Error: " << e.what() << endl;
1525                    continue;
1526                }
1527
1528
1529                if (balance >= 0) {
1530                    break;
1531                }
1532            }
1533
1534            cout << "Password: ";
1535            cin >> password;
1536
1537
1538
1539            for (Bank* bank : allBanks) {
1540                if (bank->getName() == bankName) {
1541                    bank->addAccount(Account(bankName, username, cardNumber, AccountNumber,
    password, balance));
1542                    bankFound = true;
1543                    break;
1544                }
1545            }
1546            if (!bankFound) {
1547                cout << "Bank not found. Account creation has been failed.\n";
1548            }
1549        }
1550    } while (createAccountInput == "yes");
1551
1552
1553    cout << "\n======< 'ATM Creation Step' >======\n";
1554    vector<ATM> atms;
1555    string createATMInput;
1556    int atmCount = 1;
1557    do {
1558        cout << "\nWould you like to create ATM " << atmCount << "? (yes or no) -> ";
```

```
1559        cin >> createATMInput;
1560
1561        while (createATMInput != "yes" && createATMInput != "no") {
1562            cout << "Invalid input. Please type 'yes' or 'no': ";
1563            cin >> createATMInput;
1564        }
1565
1566        if (createATMInput == "yes") {
1567            string atmType, primaryBankName, serialNumber, language;
1568            int cash1000, cash5000, cash10000, cash50000;
1569            Bank* primaryBank = nullptr;
1570
1571            cout << "Primary Bank Name: ";
1572            cin >> primaryBankName;
1573
1574            primaryBank = findBankByName(primaryBankName, allBanks);
1575
1576
1577            bool isDuplicate{ false };
1578            do {
1579                bool serial_is_6digits{ false };
1580
1581                while (!serial_is_6digits) {
1582                    cout << "Serial Number(6-digit): ";
1583                    cin >> serialNumber;
1584
1585                    if (serialNumber.length() != 6) {
1586                        cout << "Invalid serial number. It must be exactly 6 digits.\n";
1587                        continue;
1588
1589
1590                    }
1591                    else {
1592                        serial_is_6digits = true;
1593                    }
1594                }
1595
1596
1597
1598                try {
1599                    for (const ATM& atm : atms) {
1600                        if (atm.getSerialNumber() == serialNumber) {
1601                            throw runtime_error("Duplicate serial number detected.");
1602                        }
1603                    }
1604                    isDuplicate = false;
1605                }
1606                catch (const runtime_error& e) {
1607                    cout << "Error: " << e.what() << endl;
```

```
1608                    isDuplicate = true;
1609                    continue;
1610                }
1611            } while (isDuplicate);
1612
1613            do {
1614                cout << "Type(Single or Multi): ";
1615                cin >> atmType;
1616
1617                for (int i = 0; i < atmType.size(); i++) {
1618                    if ('A' <= atmType[i] && atmType[i] <= 'Z') {
1619                        atmType[i] += 32;
1620                    }
1621                }
1622                if (atmType != "single" && atmType != "multi") {
1623                    cout << "Incorrect ATM Type. Please enter within 'Single' or 'Multi'.
    \n";
1624                }
1625            } while (atmType != "single" && atmType != "multi");
1626            atmType[0] -= 32;
1627
1628            do {
1629                cout << "Language(Uni or Bi): ";
1630                cin >> language;
1631
1632                for (int i = 0; i < language.size(); i++) {
1633                    if ('A' <= language[i] && language[i] <= 'Z') {
1634                        language[i] += 32;
1635                    }
1636                }
1637                if (language != "uni" && language != "bi") {
1638                    cout << "Incorrect Language Type. Please enter within 'Uni' or 'Bi'.\n";
1639                }
1640            } while (language != "uni" && language != "bi");
1641            language[0] -= 32;
1642
1643
1644
1645            cout << "Number of initial 1,000 Cash?: ";
1646            while (!(cin >> cash1000) || cash1000 < 0) {
1647                cin.clear();
1648                cin.ignore(1000, '\n');
1649                cout << "Invalid input. Please enter a non-negative integer for 1,000 Cash:
    ";
1650            }
1651
1652            cout << "Number of initial 5,000 Cash?: ";
1653            while (!(cin >> cash5000) || cash5000 < 0) {
1654                cin.clear();
```

```
1655                cin.ignore(1000, '\n');
1656                cout << "Invalid input. Please enter a non-negative integer r for 5,000
    Cash: ";
1657            }
1658
1659            cout << "Number of initial 10,000 Cash?: ";
1660            while (!(cin >> cash10000) || cash10000 < 0) {
1661                cin.clear();
1662                cin.ignore(1000, '\n');
1663                cout << "Invalid input. Please enter a non-negative integer  for 10,000
    Cash: ";
1664            }
1665
1666            cout << "Number of initial 50,000 Cash?: ";
1667            while (!(cin >> cash50000) || cash50000 < 0) {
1668                cin.clear();
1669                cin.ignore(1000, '\n');
1670                cout << "Invalid input. Please enter a non-negative integer  for 50,000
    Cash: ";
1671            }
1672
1673
1674
1675            if (atmType == "Single") {
1676                for (Bank* bank : allBanks) {
1677                    if (bank->getName() == primaryBankName) {
1678                        primaryBank = bank;
1679                        break;
1680                    }
1681                }
1682                if (!primaryBank) {
1683                    cout << "Primary bank has not been found. Skipping ATM creation.\n";
1684                    continue;
1685                }
1686            }
1687
1688
1689            atms.emplace_back(atmType, primaryBank, serialNumber, language, cash1000,
    cash5000, cash10000, cash50000);
1690            cout << "ATM " << atmCount << " created successfully.\n";
1691
1692
1693            atmCount++;
1694        }
1695    } while (createATMInput == "yes");
1696
1697
1698
1699
1700    string command;
```

```
1701     while (true) {
1702         cout << "\nEnter '/' to view display snapshot, 'session' to start an ATM session, or
     'exit' to quit program: ";
1703         cin >> command;
1704 
1705         if (command == "/") {
1706             displaySnapshot(banks, atms);
1707         }
1708         else if (command == "session") {
1709             if (atms.empty()) {
1710                 cout << "No ATMs are available. Please create an ATM first.\n";
1711             }
1712             else {
1713                 int atmIndex;
1714                 cout << "Enter ATM index (1 to " << atms.size() << ") to start session: ";
1715                 cin >> atmIndex;
1716 
1717                 if (atmIndex >= 1 && atmIndex <= atms.size()) {
1718                     atms[atmIndex - 1].startSession(allBanks, banks, atms);
1719                 }
1720                 else {
1721                     cout << "Invalid ATM index.\n";
1722                 }
1723             }
1724         }
1725         else if (command == "exit") {
1726             break;
1727         }
1728     }
1729 
1730     return 0;
1731 }
1732 
```

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						