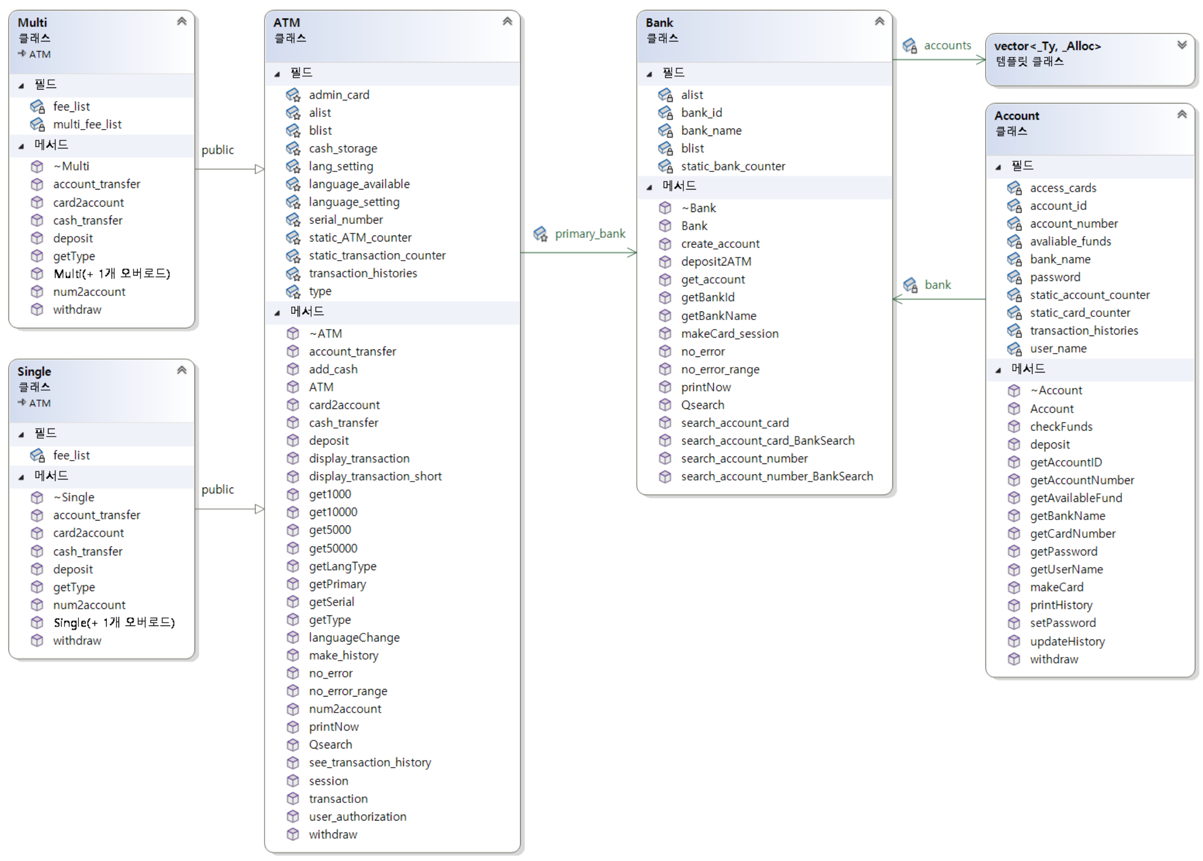
XIV. Term Project Report

201811043 김태형, 201811147 이지환, 201811195 한도희

1. Final Class Diagram Design



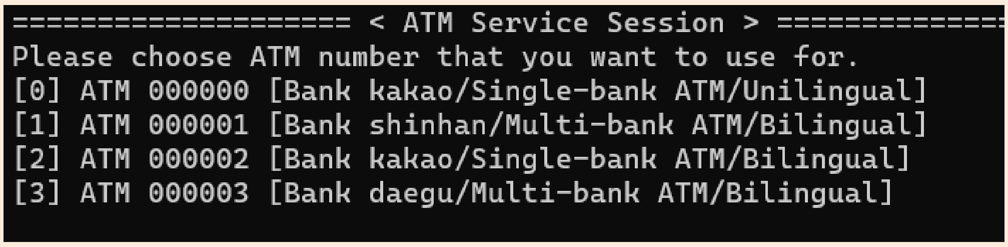
2. Requirements that have been successfully implemented

A. System Requirements

- (REQ 1.1) An ATM has a 6-digit serial number that can be uniquely identified among all ATMs (e.g., 315785).

💡 ATM을 설치할 때, 각 ATM은 6자리의 serial number를 자동으로 부여받는다. ATM이 부여받는 번호는 000000번부터 1씩 증가하여 999999번씩 부여받으므로, 1백만 대의 ATM이 개설되기 전까지는 중복이 발생하지 않는다.

아래 이미지는 4개의 ATM을 생성한 결과로, 순서대로 000000, 000001, 000002, 000003의 일련번호를 부여받았음을 확인할 수 있다.



```
==================== < ATM Service Session > =============
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank kakao/Single-bank ATM/Unilingual]
[1] ATM 000001 [Bank shinhan/Multi-bank ATM/Bilingual]
[2] ATM 000002 [Bank kakao/Single-bank ATM/Bilingual]
[3] ATM 000003 [Bank daegu/Multi-bank ATM/Bilingual]
```

- (REQ 1.2) 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 Multi-Bank ATM, there is a primary bank that manages the ATM, but a card issued by any other banks is considered valid.

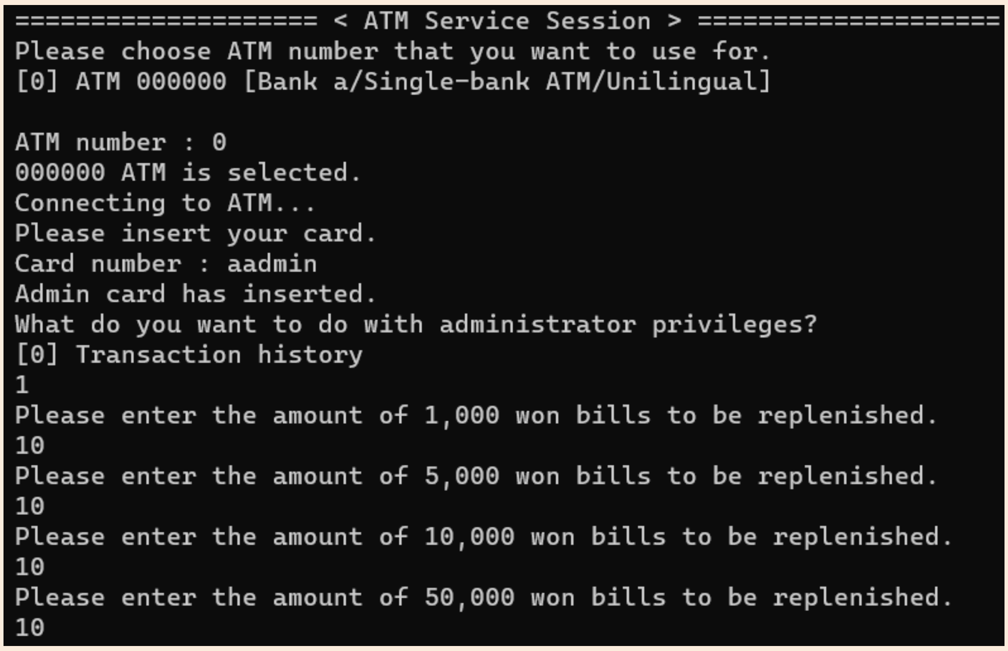
💡 class ATM은 자식 클래스로 Single Bank ATM(class Single)과 Multi Bank ATM(class Multi)를 두고 있다. Single Bank ATM은 다른 은행의 카드를 받을 경우 "지원되지 않는 카드입니다" 라는 메시지를 반환하고 세션을 종료하지만, Multi Bank ATM은 정상적으로 사용자(aaa)를 인식하고 거래를 시작한다. 아래 이미지들은 각각 kakao 은행에 생성된 과정, 그리고 daegu 은행의 single bank ATM과 multi bank ATM에 kakao 은행 card를 입력했을 때의 결과이다.

- (REQ 1.3) 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.

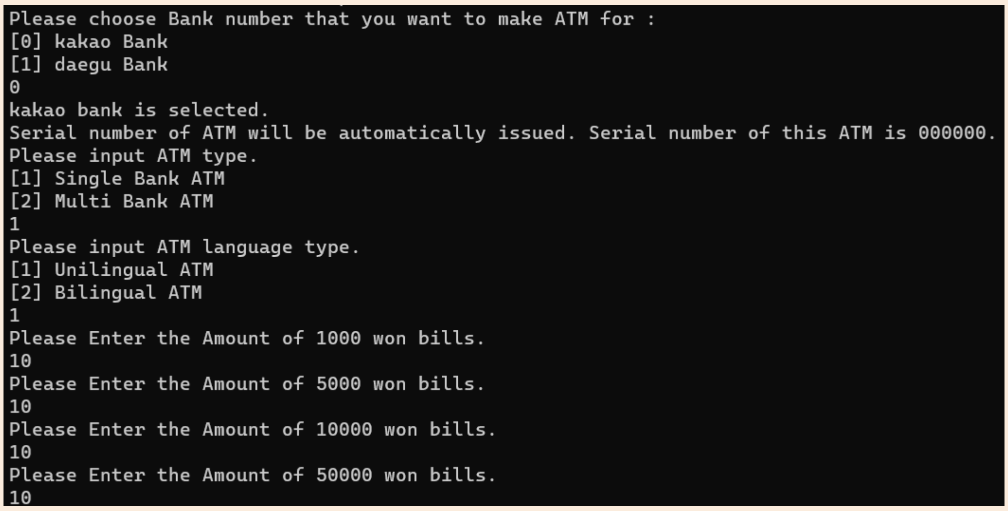
💡 각 ATM은 language type(unilingual/bilingual)을 속성으로 가지고 있다. Session이 시작되었을 때 language change 기능을 사용하면, 언어를 변경할 수 있다. 이때, unilingual ATM의 경우, “unilingual ATM cannot change language!” 라는 메시지를 출력하고 세션 시작 단계로 돌아가지만, Bilingual ATM의 경우 정상적으로 언어를 선택할 수 있다. 아래 메시지는 각각 unilingual/bilingual ATM에서 언어 변경 기능을 사용했을 때의 결과이다.

- (REQ 1.4) A Bank deposits a certain amount of cashes to an ATM to serve users.

💡 ATM을 초기에 개설할 때 Bank를 선택하여, ATM이 보유한 초기 현금을 설정할 수 있다. 추가로, ATM에 접속하여 admin card를 입력한 후 숨겨진 명령어(1)를 입력하면 ATM에 현금을 보충할 수 있다. 아래 이미지들은 각각 ATM에 초기 현금을 설정하는 과정과, admin card로 현금을 보충하는 과정이다.



```
=================== < ATM Service Session > ===================
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank a/Single-bank ATM/Unilingual]

ATM number : 0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : aadmin
Admin card has inserted.
What do you want to do with administrator privileges?
[0] Transaction history
1
Please enter the amount of 1,000 won bills to be replenished.
10
Please enter the amount of 5,000 won bills to be replenished.
10
Please enter the amount of 10,000 won bills to be replenished.
10
Please enter the amount of 50,000 won bills to be replenished.
10
```



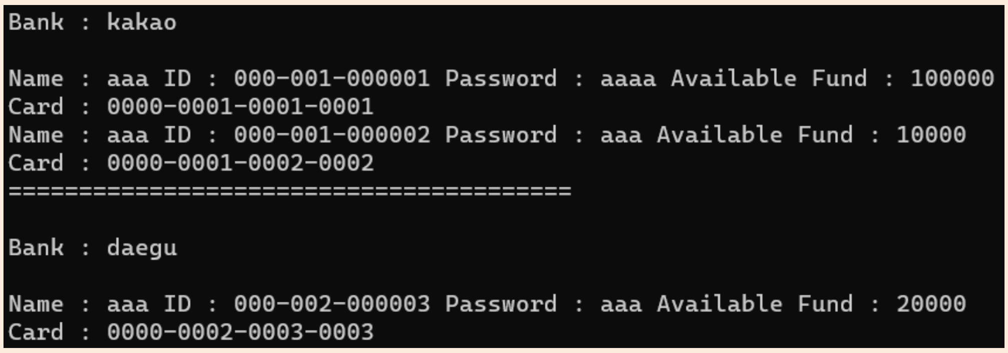
```
Please choose Bank number that you want to make ATM for :
[0] kakao Bank
[1] daegu Bank
0
kakao bank is selected.
Serial number of ATM will be automatically issued. Serial number of this ATM is 000000.
Please input ATM type.
[1] Single Bank ATM
[2] Multi Bank ATM
1
Please input ATM language type.
[1] Unilingual ATM
[2] Bilingual ATM
1
Please Enter the Amount of 1000 won bills.
10
Please Enter the Amount of 5000 won bills.
10
Please Enter the Amount of 10000 won bills.
10
Please Enter the Amount of 50000 won bills.
10
```

- (REQ 1.5) A Bank can open an Account for user with necessary information to perform bank services.
    - (e.g.) Bank name (e.g, Kakao, Shinhan), User name, Account number (12-digit), Available funds, Transaction histories.

💡 Bank가 존재하면, 사용자는 해당 bank에 접속하여 account를 개설할 수 있다. Account는 접속한 Bank에 개설되며, user name과 password, availabe funds를 수동으로 입력받는다. Account number는 자동으로 부여받으며, transaction history는 거래를 진행하며 입력된다. 이 정보들은 Bank admin session에서 확인할 수 있다.

- (REQ 1.6) A user may have multiple Accounts in a Bank.

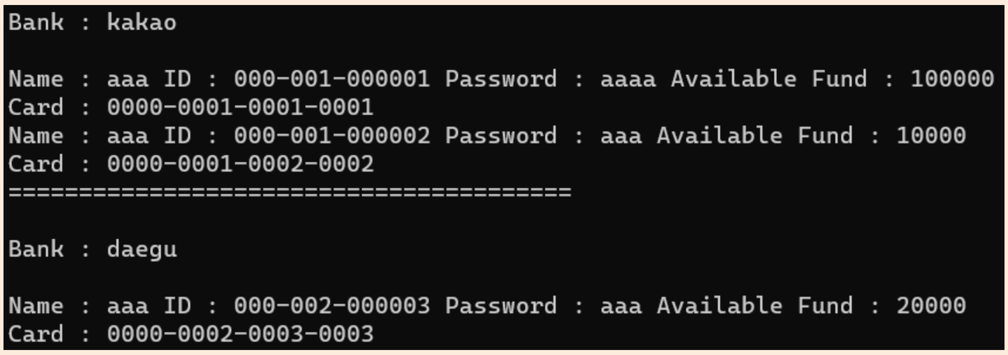
💡 사용자는 한 은행에 여러 개의 계좌를 만들 수 있다. 아래 이미지는 사용자 aaa가 kakao 은행에 여러 계좌를 만든 예시이다.



```
Bank : kakao

Name : aaa ID : 000-001-000001 Password : aaaa Available Fund : 100000
Card : 0000-0001-0001-0001
Name : aaa ID : 000-001-000002 Password : aaa Available Fund : 10000
Card : 0000-0001-0002-0002
========================================

Bank : daegu

Name : aaa ID : 000-002-000003 Password : aaa Available Fund : 20000
Card : 0000-0002-0003-0003
```

- (REQ 1.7) A user may have Accounts in multiple Banks.

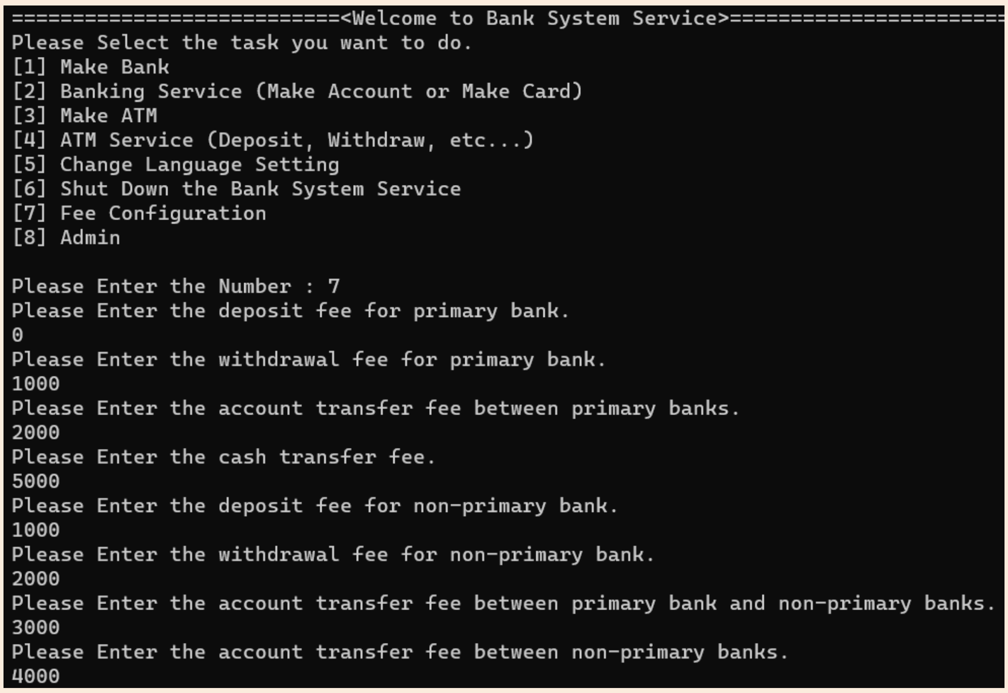
💡 사용자는 여러 은행에 계좌를 만들 수 있다. 아래 이미지는 사용자 aaa가 kakao, daegu 은행에 각각 계좌를 만든 예시이다.



```
Bank : kakao

Name : aaa ID : 000-001-000001 Password : aaaa Available Fund : 100000
Card : 0000-0001-0001-0001
Name : aaa ID : 000-001-000002 Password : aaa Available Fund : 10000
Card : 0000-0001-0002-0002
========================================

Bank : daegu

Name : aaa ID : 000-002-000003 Password : aaa Available Fund : 20000
Card : 0000-0002-0003-0003
```

- (REQ 1.8) Each ATM have several types of transaction fees, and paid as follows: (후략)

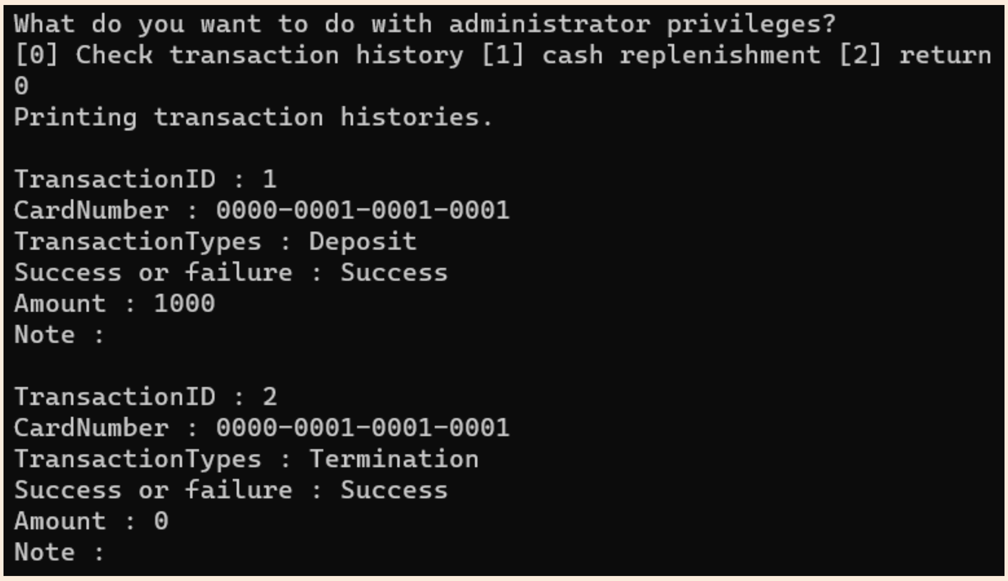
💡 콘솔에서 Fee의 각 항목을 능동적으로 초기화하거나 설정할 수 있다. 각 Fee는 포인터 형식으로 heap에 저장되어, 각 ATM에 전달된다(Fee 설정을 변경하면 모든 ATM에 즉각적으로 반영된다).



```
===========================<Welcome to Bank System Service>======================
Please Select the task you want to do.
[1] Make Bank
[2] Banking Service (Make Account or Make Card)
[3] Make ATM
[4] ATM Service (Deposit, Withdraw, etc...)
[5] Change Language Setting
[6] Shut Down the Bank System Service
[7] Fee Configuration
[8] Admin

Please Enter the Number : 7
Please Enter the deposit fee for primary bank.
0
Please Enter the withdrawal fee for primary bank.
1000
Please Enter the account transfer fee between primary banks.
2000
Please Enter the cash transfer fee.
5000
Please Enter the deposit fee for non-primary bank.
1000
Please Enter the withdrawal fee for non-primary bank.
2000
Please Enter the account transfer fee between primary bank and non-primary banks.
3000
Please Enter the account transfer fee between non-primary banks.
4000
```

- (REQ 1.9) An admin can access the menu of “Transaction History” via an admin card (See REQ Display of Transaction History).

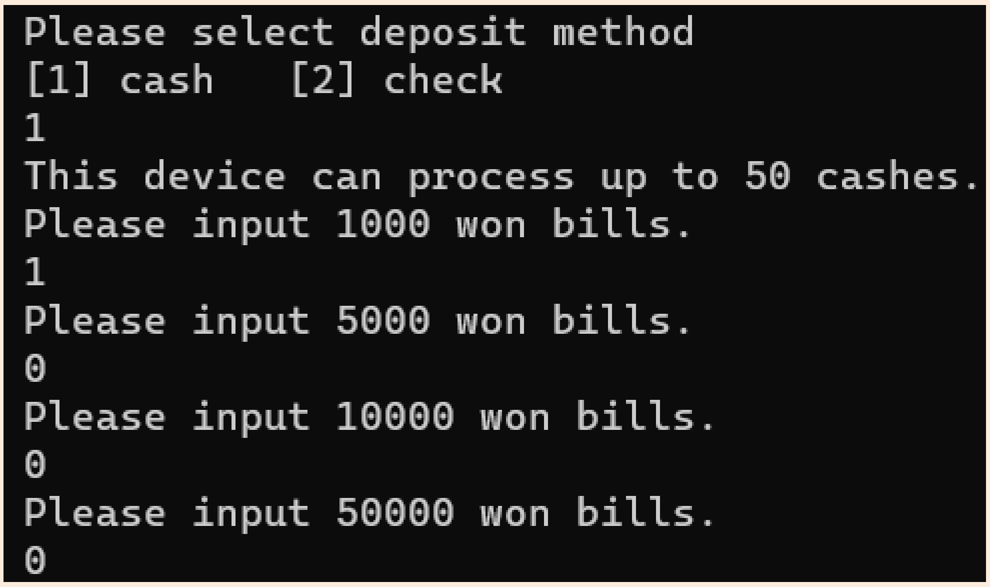
💡 admin session으로 진입하면 Check transaction history 항목을 확인할 수 있다. 해당 항목으로 진입하면 Transaction history 기능에 접속할 수 있다.



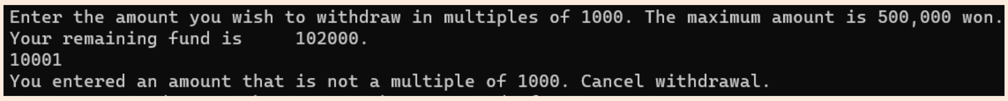
```
What do you want to do with administrator privileges?
[0] Check transaction history [1] cash replenishment [2] return
0
Printing transaction histories.

TransactionID : 1
CardNumber : 0000-0001-0001-0001
TransactionTypes : Deposit
Success or failure : Success
Amount : 1000
Note :

TransactionID : 2
CardNumber : 0000-0001-0001-0001
TransactionTypes : Termination
Success or failure : Success
Amount : 0
Note :
```

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

ATM은 내부적으로 1000원, 5000원, 10000원, 50000원 지폐의 장수를 저장하는 속성을 가지고 있으며, 현금 거래는 해당 속성만을 이용해 진행된다. 입금은 임의의 액수 대신 각 지폐의 장수를 받으며, 출금 액수가 1000의 배수가 아닐 경우 올바르지 않은 입력으로 처리되어 출금이 취소된다.

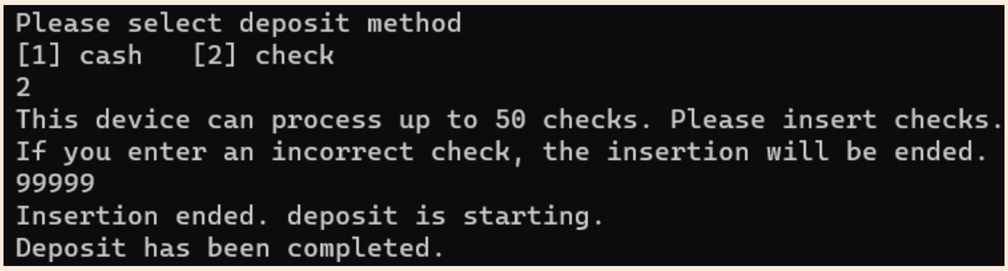


```
Please select deposit method
[1] cash    [2] check
1
This device can process up to 50 cashes.
Please input 1000 won bills.
1
Please input 5000 won bills.
0
Please input 10000 won bills.
0
Please input 50000 won bills.
0
```



```
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is     102000.
10001
You entered an amount that is not a multiple of 1000. Cancel withdrawal.
```

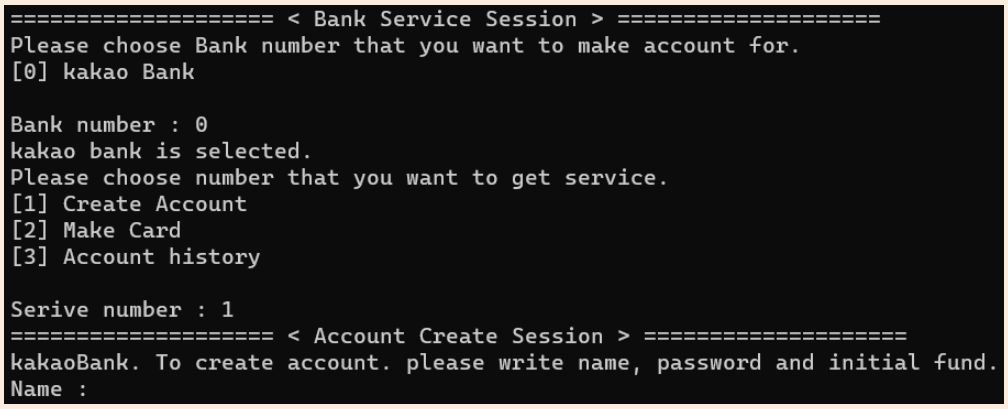
수표는 임의의 액수를 입력받을 수 있으나, 100000원보다 작은 금액을 입력받을 경우 수표 입력을 중단하고 현재까지 입력받은 수표 가격의 합계를 계산한다(이미지에서는 99999원의 수표를 입력받기 전에 올바른 수표가 한 장도 입력되지 않았으므로, 0원을 입금한 것으로 처리되었다).



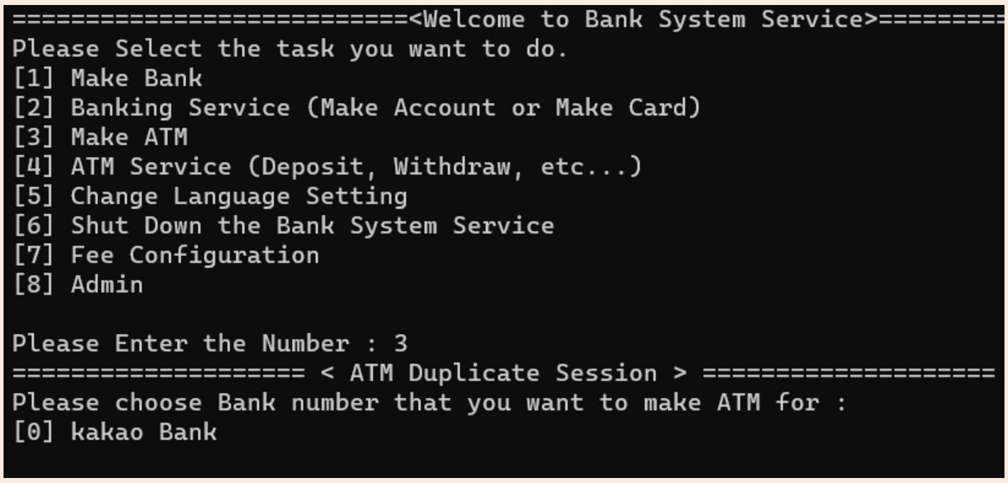
```
Please select deposit method
[1] cash    [2] check
2
This device can process up to 50 checks. Please insert checks.
If you enter an incorrect check, the insertion will be ended.
99999
Insertion ended. deposit is starting.
Deposit has been completed.
```

- (REQ 1.11) All accounts and ATMs shall be created and initialized during the program execution.

💡 Account와 ATM은 모두 프로그램 실행 후에 생성되어, 필수적인 정보를 입력받거나 내부 method에 의거하여 자동으로 부여받는다(우리는 Bank 또한 프로그램 실행 후에 생성되도록 구현하였다). 아래 이미지들은 Account와 ATM을 생성할 때 정보들을 입력받는 과정이다.



```
==================== < Bank Service Session > ====================
Please choose Bank number that you want to make account for.
[0] kakao Bank

Bank number : 0
kakao bank is selected.
Please choose number that you want to get service.
[1] Create Account
[2] Make Card
[3] Account history

Serive number : 1
==================== < Account Create Session > ====================
kakaoBank. To create account. please write name, password and initial fund.
Name :
```

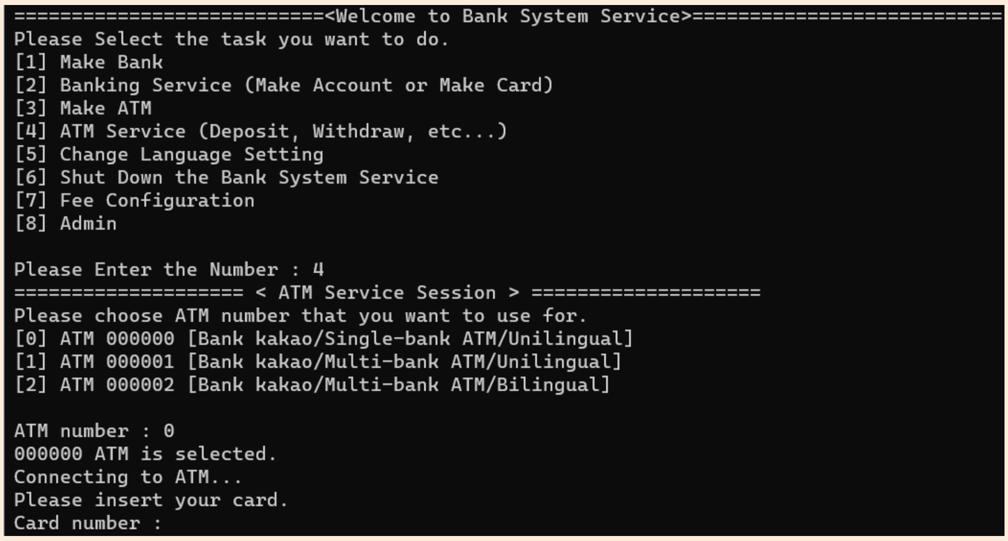


```
===========================<Welcome to Bank System Service>=========
Please Select the task you want to do.
[1] Make Bank
[2] Banking Service (Make Account or Make Card)
[3] Make ATM
[4] ATM Service (Deposit, Withdraw, etc...)
[5] Change Language Setting
[6] Shut Down the Bank System Service
[7] Fee Configuration
[8] Admin

Please Enter the Number : 3
==================== < ATM Duplicate Session > ====================
Please choose Bank number that you want to make ATM for :
[0] kakao Bank
```

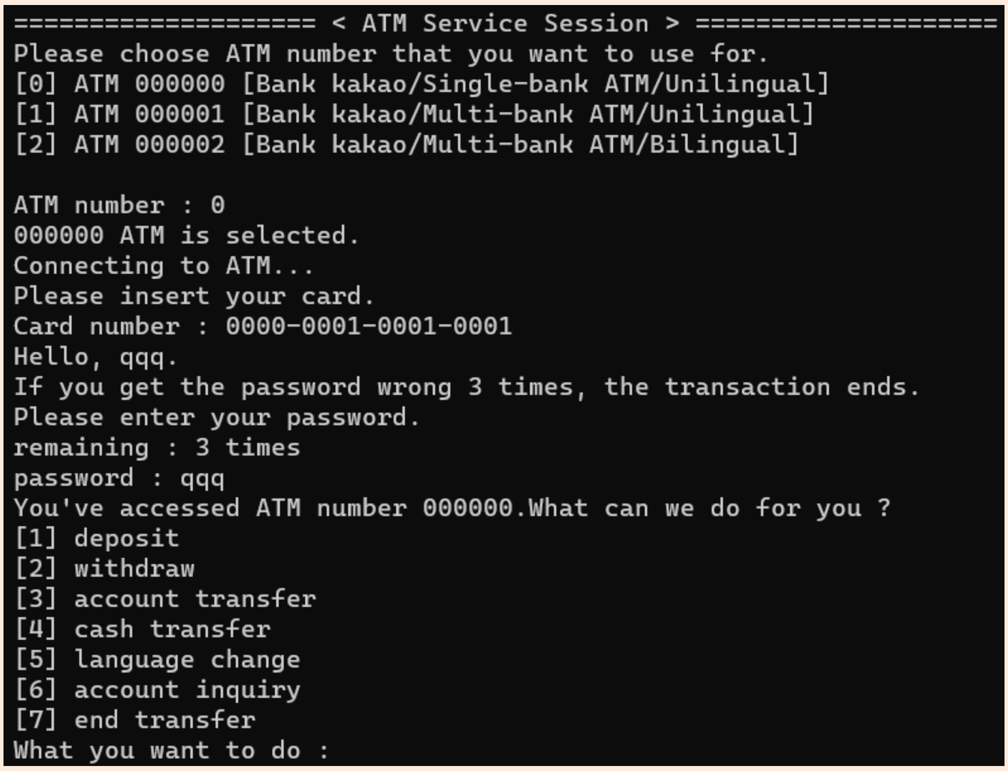
B. ATM Session Requirements

- (REQ 2.1) A session starts when a user inserts a card.

💡 초기 화면에서 ATM Service 항목에 접속하고 사용할 ATM을 선택하면, 카드 번호를 입력받아야 한다. 카드 번호와 비밀번호를 제대로 입력받으면, 세션이 시작되어 거래를 진행할 수 있다. 아래 이미지들은 각각 사용할 ATM의 선택과, 카드 번호를 입력받아 세션을 시작하는 장면이다.



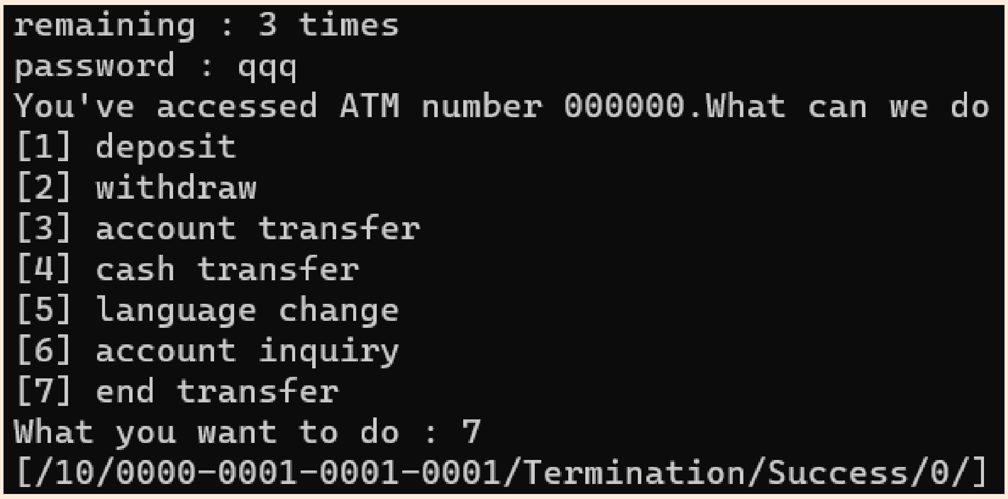
```
===========================<Welcome to Bank System Service>===========================
Please Select the task you want to do.
[1] Make Bank
[2] Banking Service (Make Account or Make Card)
[3] Make ATM
[4] ATM Service (Deposit, Withdraw, etc...)
[5] Change Language Setting
[6] Shut Down the Bank System Service
[7] Fee Configuration
[8] Admin

Please Enter the Number : 4
==================== < ATM Service Session > ====================
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank kakao/Single-bank ATM/Unilingual]
[1] ATM 000001 [Bank kakao/Multi-bank ATM/Unilingual]
[2] ATM 000002 [Bank kakao/Multi-bank ATM/Bilingual]

ATM number : 0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number :
```



```
==================== < ATM Service Session > ====================
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank kakao/Single-bank ATM/Unilingual]
[1] ATM 000001 [Bank kakao/Multi-bank ATM/Unilingual]
[2] ATM 000002 [Bank kakao/Multi-bank ATM/Bilingual]

ATM number : 0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : 0000-0001-0001-0001
Hello, qqq.
If you get the password wrong 3 times, the transaction ends.
Please enter your password.
remaining : 3 times
password : qqq
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do :
```

- (REQ 2.2) 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).

💡 거래 선택 화면에서, 7을 입력하여 end transfer 기능을 선택하면 세션이 종료된다. 혹은, 출금 시 잔액이 부족한 등 다양한 예외 사항이 발생하면 세션이 종료된다. 이하의 이미지는 유저가 직접 세션을 종료하도록 하였을 때이다.



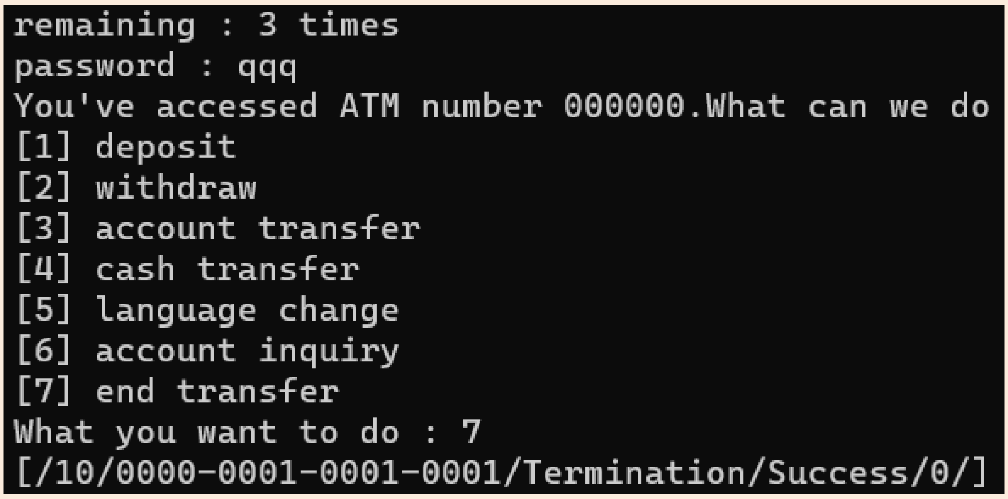
```
remaining : 3 times
password : qqq
You've accessed ATM number 000000.What can we do
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 7
[/10/0000-0001-0001-0001/Termination/Success/0/]
```

- (REQ 2.3) When a session ends, the summary of all transactions performed in a session must be displayed.

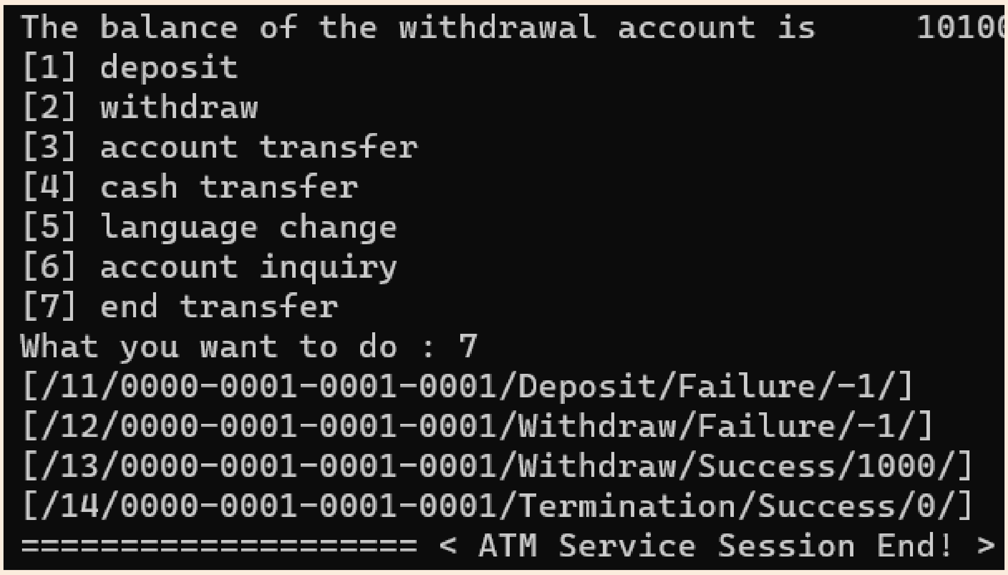
💡 세션을 종료하면, 간략한 거래 정보가 표시된다.

표시되는 거래의 종류는 입금/출금/계좌 송금/현금 송금/언어 변경/계좌 조회/세션 종료이다. 언어 변경과 계좌 조회, 세션 종료는 엄밀히 말해 거래가 아니지만, 그 내역 또한 출력할 수 있게 구현하였다.

각 거래에 대해 거래 일련번호/카드 번호/거래 종류/성공 여부/액수(언어 변경 및 계좌 조회는 0, 실패 시에는 -1)/개별 정보(송금받는 계좌 정보 등)가 출력된다.



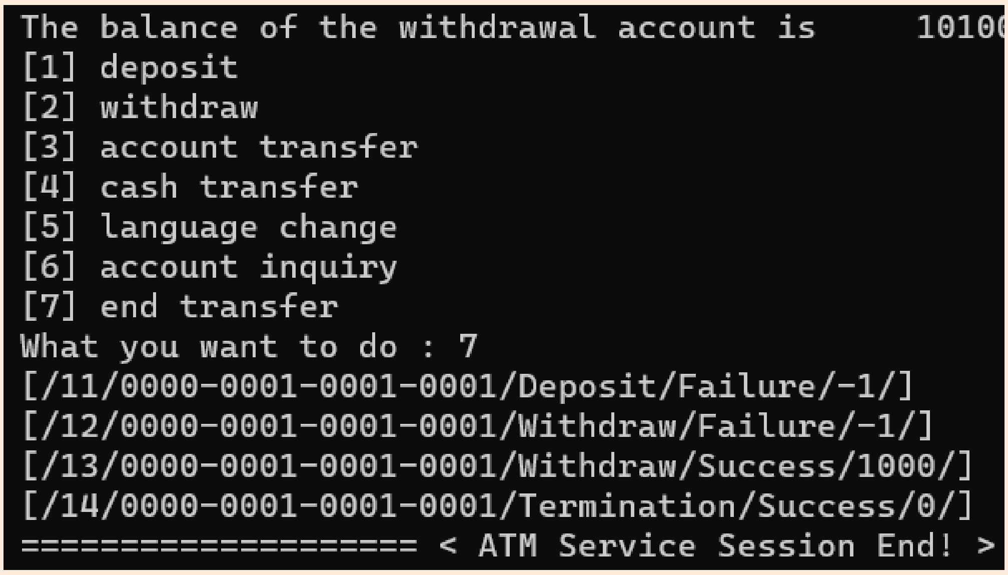
```
remaining : 3 times
password : qqq
You've accessed ATM number 000000.What can we do
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 7
[/10/0000-0001-0001-0001/Termination/Success/0/]
```



```
 The balance of the withdrawal account is     10100
 [1] deposit
 [2] withdraw
 [3] account transfer
 [4] cash transfer
 [5] language change
 [6] account inquiry
 [7] end transfer
 What you want to do : 7
 [/11/0000-0001-0001-0001/Deposit/Failure/-1/]
 [/12/0000-0001-0001-0001/Withdraw/Failure/-1/]
 [/13/0000-0001-0001-0001/Withdraw/Success/1000/]
 [/14/0000-0001-0001-0001/Termination/Success/0/]
 ==================== < ATM Service Session End! >
```

- (REQ 2.4) Each transaction has a unique identifier across all sessions.

💡 거래가 진행될 때마다, 각 거래는 고유한 일련번호를 입력받는다. 각 거래의 일련번호는 0번부터 시작하여, 프로그램이 실행되는 내내 유지된다. 이하의 이미지는 이전 세션들에서 총 10번의 거래가 진행되고 난 다음 실행된 세션이 종료될 때의 출력으로, 따라서 거래들의 일련번호는 11부터 시작한다.

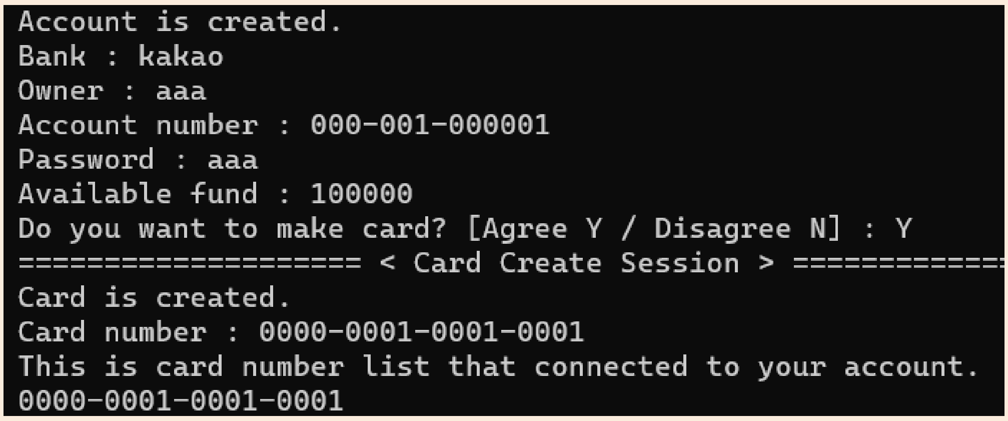


```
The balance of the withdrawal account is     10100
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 7
[/11/0000-0001-0001-0001/Deposit/Failure/-1/]
[/12/0000-0001-0001-0001/Withdraw/Failure/-1/]
[/13/0000-0001-0001-0001/Withdraw/Success/1000/]
[/14/0000-0001-0001-0001/Termination/Success/0/]
==================== < ATM Service Session End! >
```

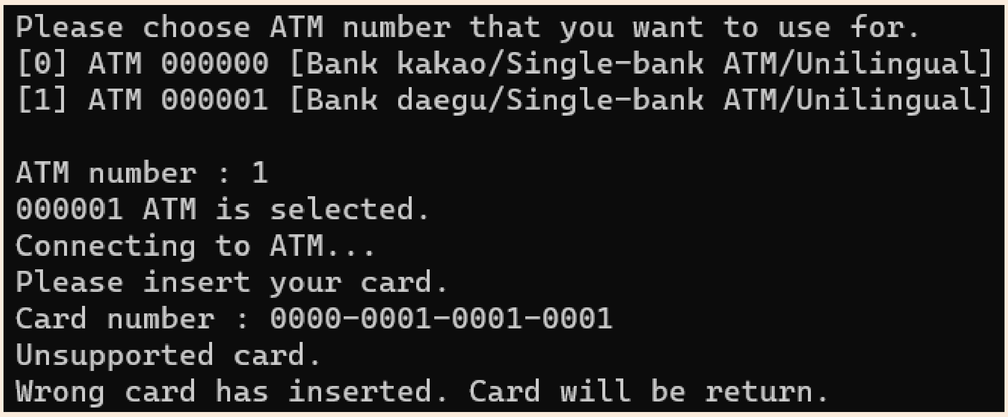
C. User Authorization Requirements

- (REQ 3.1) An ATM checks if the inserted card is valid for the current type of ATM

💡 각 ATM이 카드를 입력(입력은 번호로 받는다)받으면 은행에서 카드 번호를 검색하여, Single Bank ATM에 다른 은행의 카드가 입력되면, 오류 메시지를 출력하고 ATM 세션을 종료한다. 이하의 이미지는, kakao bank의 카드를 daegu bank의 single bank ATM에 입력한 결과이다.



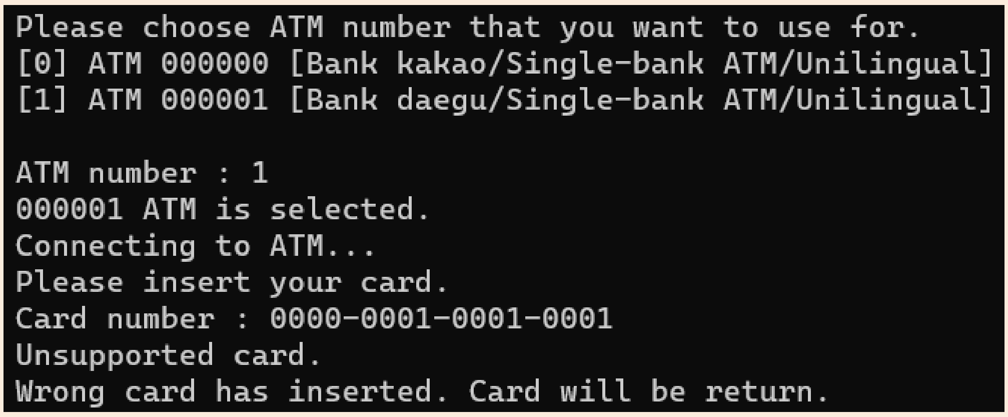
```
Account is created.
Bank : kakao
Owner : aaa
Account number : 000-001-000001
Password : aaa
Available fund : 100000
Do you want to make card? [Agree Y / Disagree N] : Y
==================== < Card Create Session > ============
Card is created.
Card number : 0000-0001-0001-0001
This is card number list that connected to your account.
0000-0001-0001-0001
```



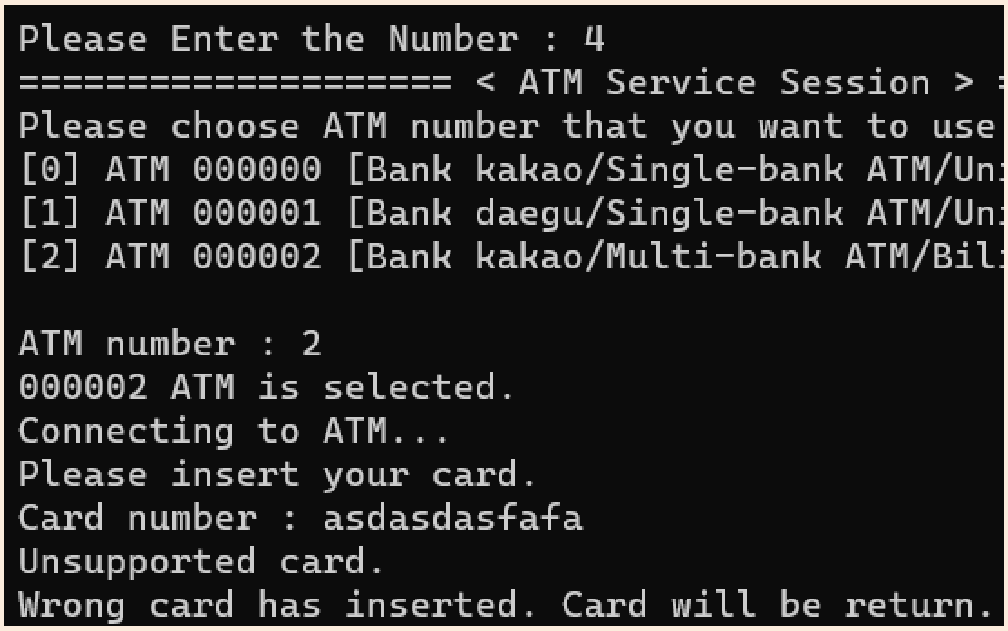
```
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank kakao/Single-bank ATM/Unilingual]
[1] ATM 000001 [Bank daegu/Single-bank ATM/Unilingual]

ATM number : 1
000001 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : 0000-0001-0001-0001
Unsupported card.
Wrong card has inserted. Card will be return.
```

- (REQ 3.2) If an invalid card is inserted, the ATM shall display an appropriate error message (e.g, Invalid Card).

💡 Single Bank ATM에 다른 은행의 카드가 삽입되거나, ATM의 type을 막론하고 존재하지 않는 카드가 삽입되었을 경우, 오류 메시지를 출력하고 ATM 세션을 종료한다. 이하의 이미지는 각각 Single bank ATM에 다른 은행의 카드를 넣었을 때와, Multi bank ATM에 존재하지 않는 카드를 넣었을 때이다.



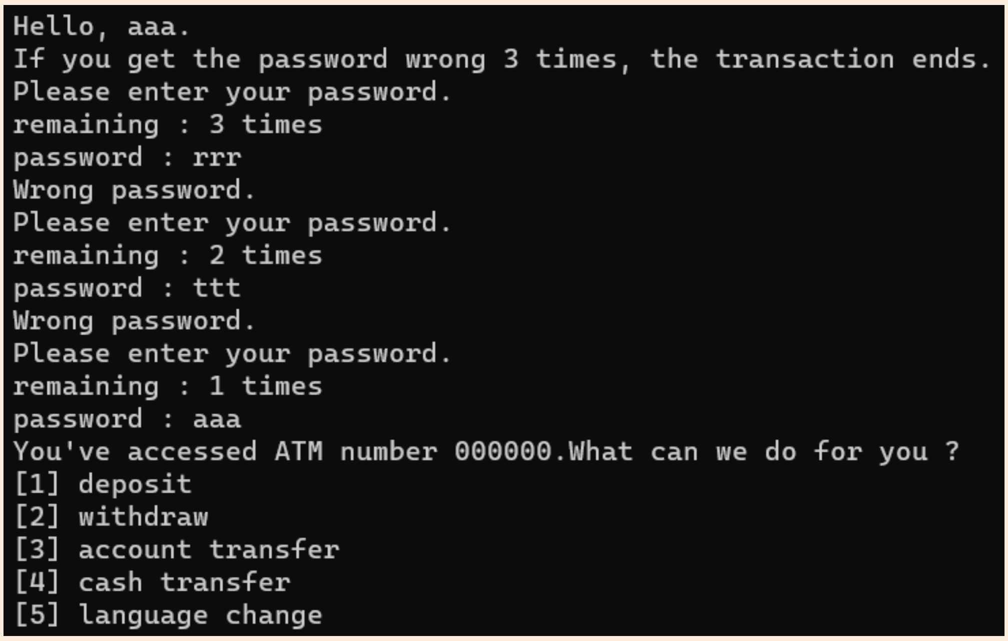
```
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank kakao/Single-bank ATM/Unilingual]
[1] ATM 000001 [Bank daegu/Single-bank ATM/Unilingual]

ATM number : 1
000001 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : 0000-0001-0001-0001
Unsupported card.
Wrong card has inserted. Card will be return.
```



```
Please Enter the Number : 4
==================== < ATM Service Session > =
Please choose ATM number that you want to use
[0] ATM 000000 [Bank kakao/Single-bank ATM/Un
[1] ATM 000001 [Bank daegu/Single-bank ATM/Un
[2] ATM 000002 [Bank kakao/Multi-bank ATM/Bil

ATM number : 2
000002 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : asdasdasfafa
Unsupported card.
Wrong card has inserted. Card will be return.
```

- (REQ 3.3) An ATM shall ask a user to enter the password (e.g., Enter Password), and verify if the password is correct.

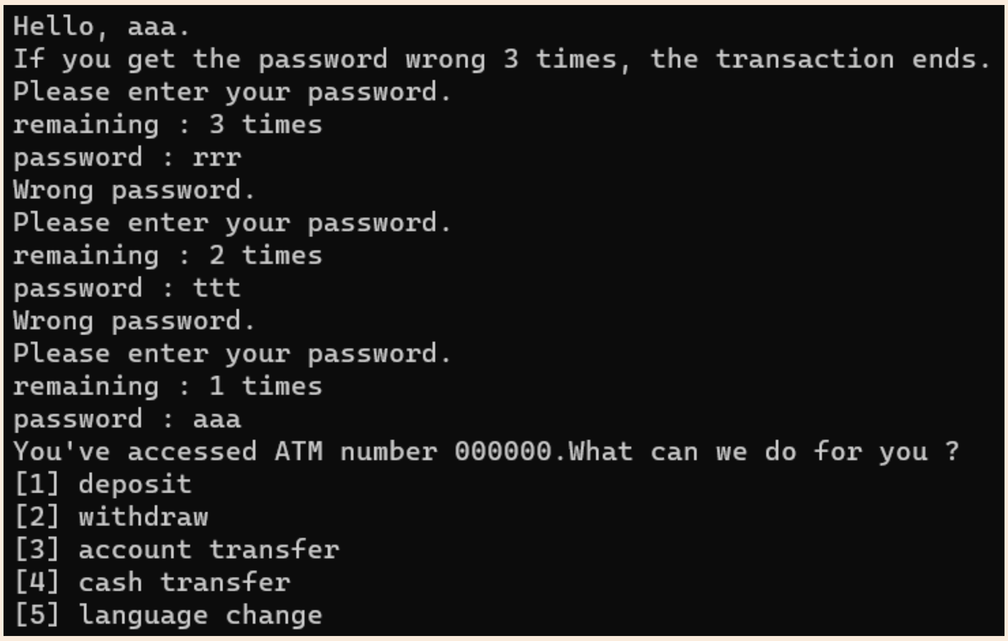
💡 ATM은 비밀번호를 입력받아, 비밀번호가 올바른지 검사한다(이 예시에서 비밀번호는 aaa이다). 비밀번호가 올바른 경우에만 세션이 시작된다. 내부적으로 ATM은 account 정보를 입력받지 않으므로, primary bank 및 bank list(Multi Bank ATM일 경우에만)에 접속하여 account 정보를 검색한다.



```
Hello, aaa.
If you get the password wrong 3 times, the transaction ends.
Please enter your password.
remaining : 3 times
password : rrr
Wrong password.
Please enter your password.
remaining : 2 times
password : ttt
Wrong password.
Please enter your password.
remaining : 1 times
password : aaa
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
```

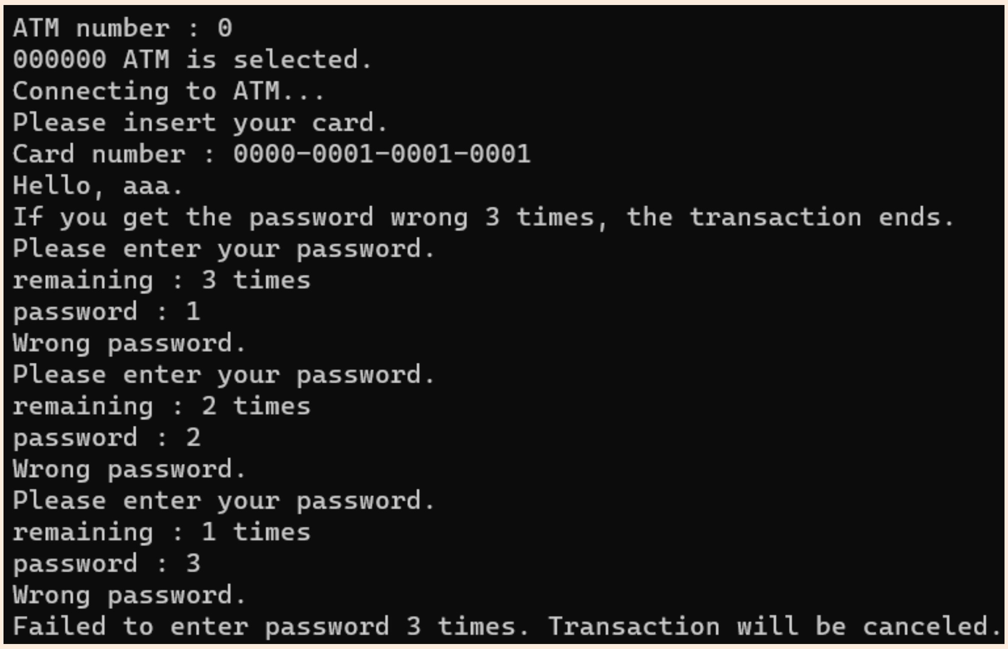
- (REQ 3.4) If the entered password is incorrect, the ATM shall display an appropriate error message(e.g., Wrong Password).

💡 비밀번호가 틀렸을 경우, “Wrong password.” 라는 오류 메시지를 출력한다. 이하의 이미지는 고의로 비밀번호를 틀렸을 때의 예시이다. 이 예시에서 비밀번호는 aaa로, rrr과 ttt라는 틀린 비밀번호가 입력되었을 때 오류 메시지가 출력되었다.


```
Hello, aaa.
If you get the password wrong 3 times, the transaction ends.
Please enter your password.
remaining : 3 times
password : rrr
Wrong password.
Please enter your password.
remaining : 2 times
password : ttt
Wrong password.
Please enter your password.
remaining : 1 times
password : aaa
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
```

- (REQ 3.5) If a user enters wrong passwords 3 times in a row, a session is aborted, and return the card to the user.

💡 비밀번호를 3번 연속으로 틀렸을 경우, 세션을 종료하고 카드를 반환한다. 이하의 이미지는 고의로 비밀번호를 3번 연속 틀렸을 때의 예시이다. 이 예시에서 카드 0000-0001-0001-0001의 비밀번호는 aaa로, 비밀번호를 3번 연속으로 틀려 세션이 종료되었다.

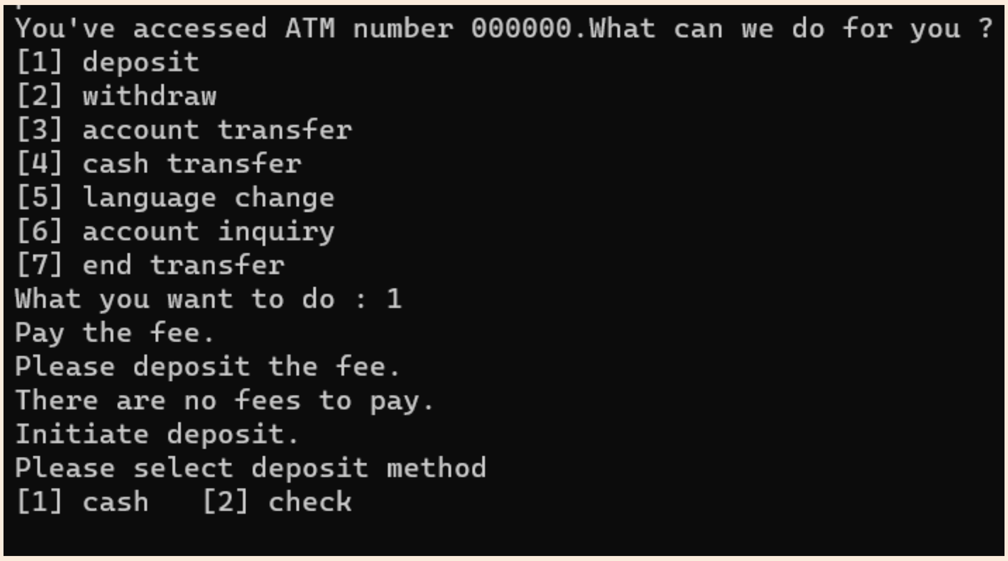


```
ATM number : 0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : 0000-0001-0001-0001
Hello, aaa.
If you get the password wrong 3 times, the transaction ends.
Please enter your password.
remaining : 3 times
password : 1
Wrong password.
Please enter your password.
remaining : 2 times
password : 2
Wrong password.
Please enter your password.
remaining : 1 times
password : 3
Wrong password.
Failed to enter password 3 times. Transaction will be canceled.
```

D. Deposit Requirements

- (REQ 4.1) An ATM shall take either cash or check from a user.

💡 입금 메뉴를 실행할 경우, ATM은 현금과 수표 중 어떤 방법으로 입금을 진행할 것인지 질문한다. 그 후, 사용자가 입력한 방법에 따라 입금을 진행한다.

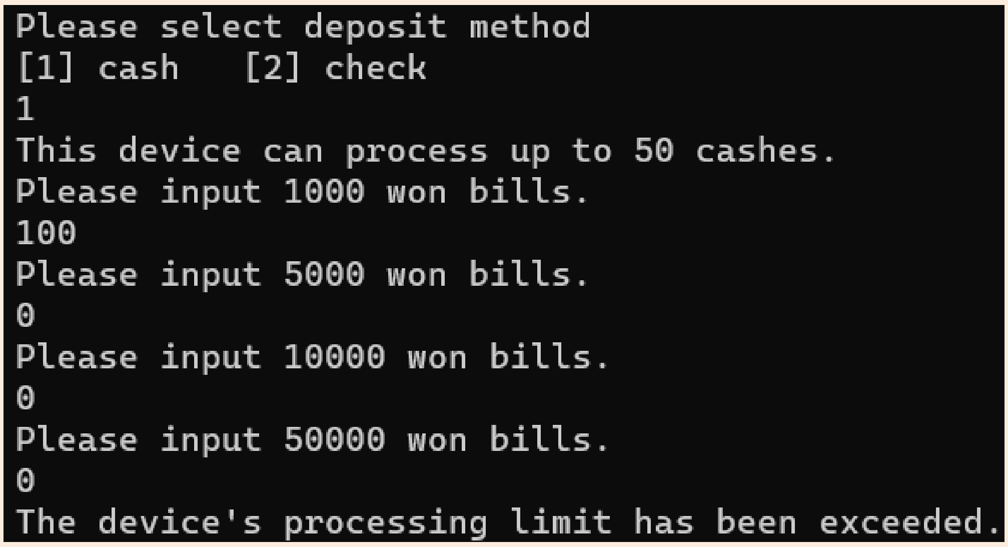


```
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 1
Pay the fee.
Please deposit the fee.
There are no fees to pay.
Initiate deposit.
Please select deposit method
[1] cash    [2] check
```

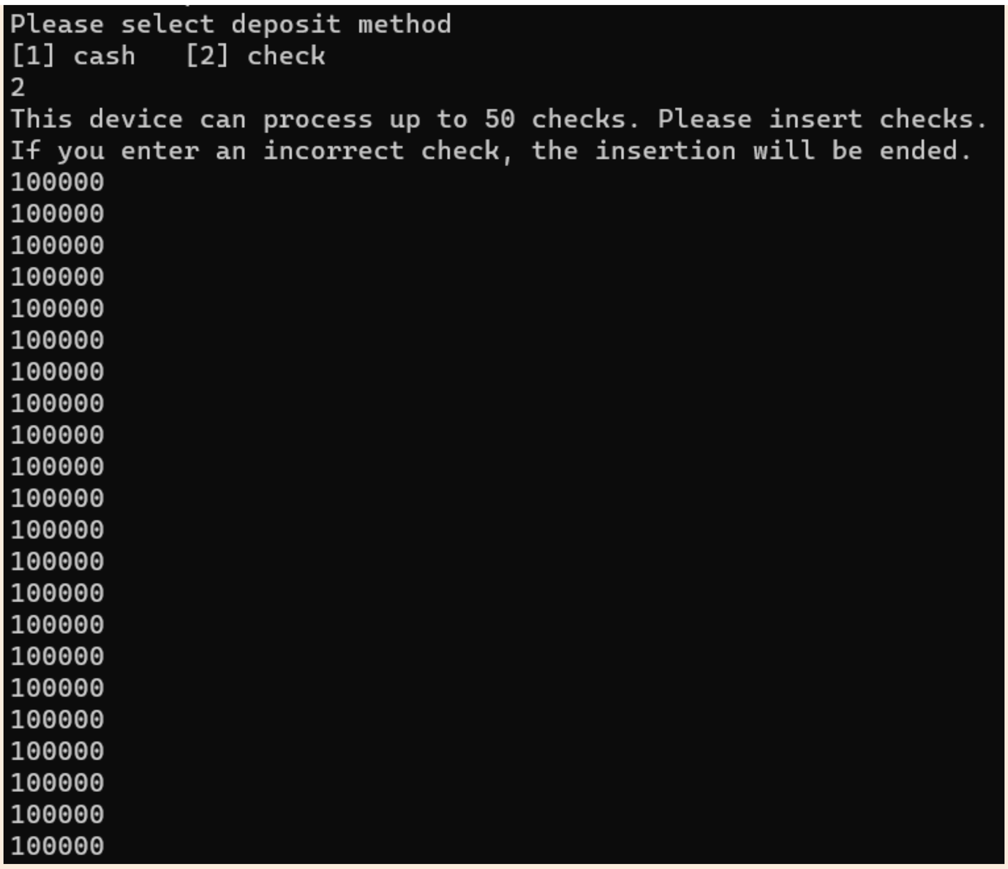
- (REQ 4.2) An ATM shall display an appropriate error message if the number of the inserted cash or checks exceed the limit allowed by the ATM.

💡 ATM이 현금을 50장 이상 입력받으면, 기계의 처리 한계를 벗어났다는 메시지를 출력한다. 수표의 경우 (100000원 수표를 일일이 입금하고 있는 아래의 이미지와 같이) 반드시 1장씩 입력받으며, 50장째의 수표가 입력되면 입력을 중단하고 즉시 입금을 진행하므로 error가 발생하지 않는다. 과제 pdf 파일의 3페이지에서는 수표 처리 한계의 예시를 50장으로 제시하였고, 6페이지에서는 30장으로 제시하여, 임의대로 50장까지 처리할 수 있도록 설정하였다.

이하의 이미지는 현금을 한계 이상으로 입력받았을 때의 오류 메시지, 그리고 수표를 1장씩 입력받는 과정이다.

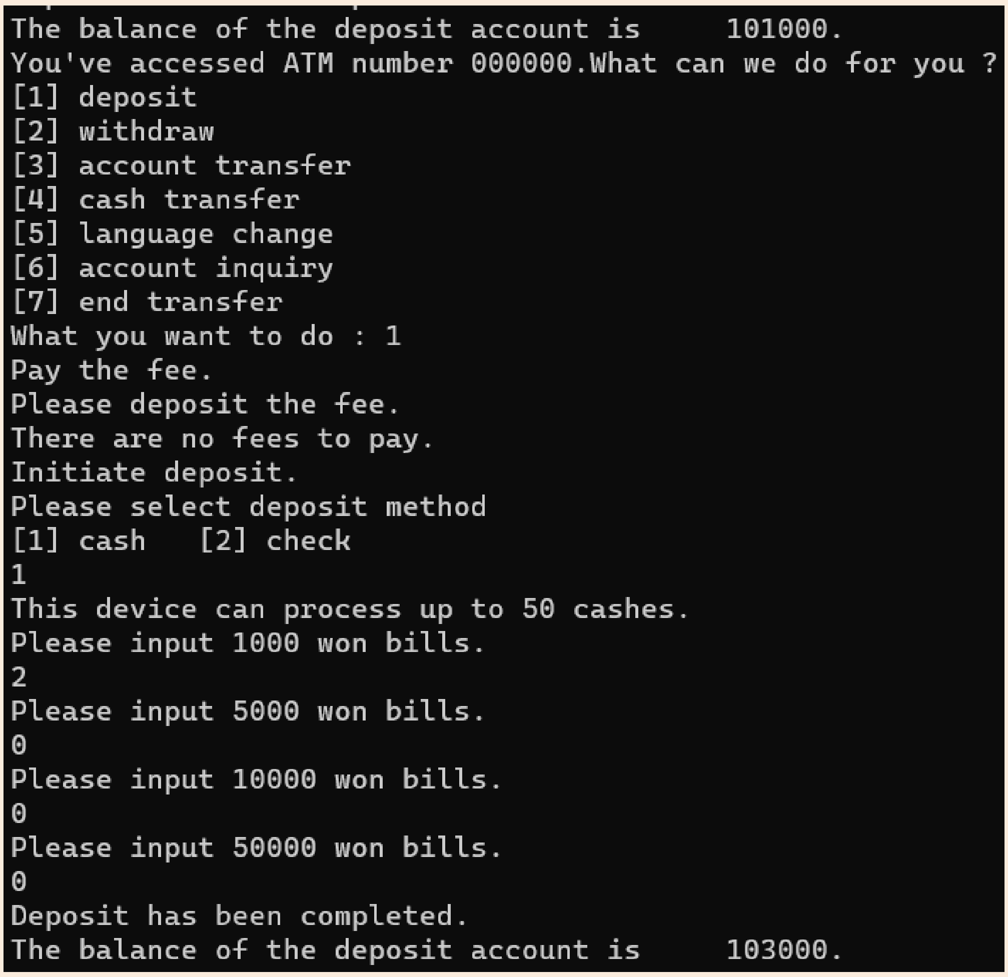


```
Please select deposit method
[1] cash   [2] check
1
This device can process up to 50 cashes.
Please input 1000 won bills.
100
Please input 5000 won bills.
0
Please input 10000 won bills.
0
Please input 50000 won bills.
0
The device's processing limit has been exceeded.
```


```
Please select deposit method
[1] cash   [2] check
2
This device can process up to 50 checks. Please insert checks.
If you enter an incorrect check, the insertion will be ended.
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
```

- (REQ 4.3) 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).

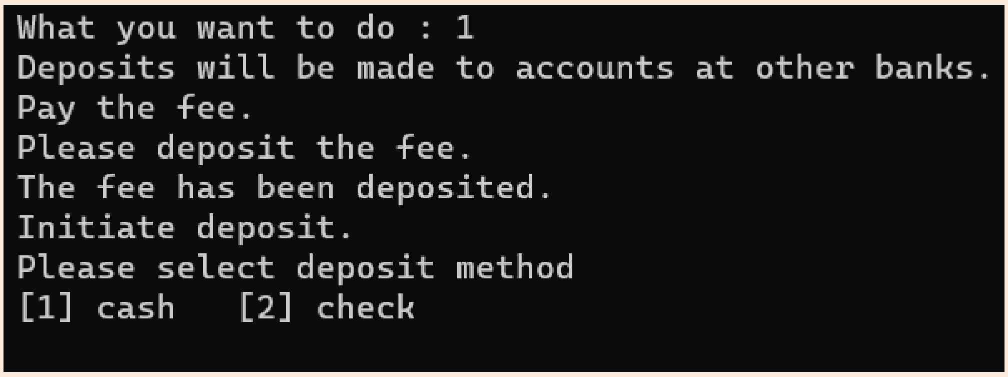
💡 입금이 완료되면, 계좌에 동일한 금액이 추가된다. 입금이 완료된 계좌의 금액은 거래가 끝난 직후 콘솔에 표기된다. 아래 이미지는 잔고가 101000원인 계좌에 2000원을 입금했을 경우의 이미지로, 101000원이었던 잔고가 103000원이 됨을 확인할 수 있다.



```
The balance of the deposit account is     101000.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 1
Pay the fee.
Please deposit the fee.
There are no fees to pay.
Initiate deposit.
Please select deposit method
[1] cash   [2] check
1
This device can process up to 50 cashes.
Please input 1000 won bills.
2
Please input 5000 won bills.
0
Please input 10000 won bills.
0
Please input 50000 won bills.
0
Deposit has been completed.
The balance of the deposit account is     103000.
```

- (REQ 4.4) Some deposit fee may be charged (See REQ in System Setup).

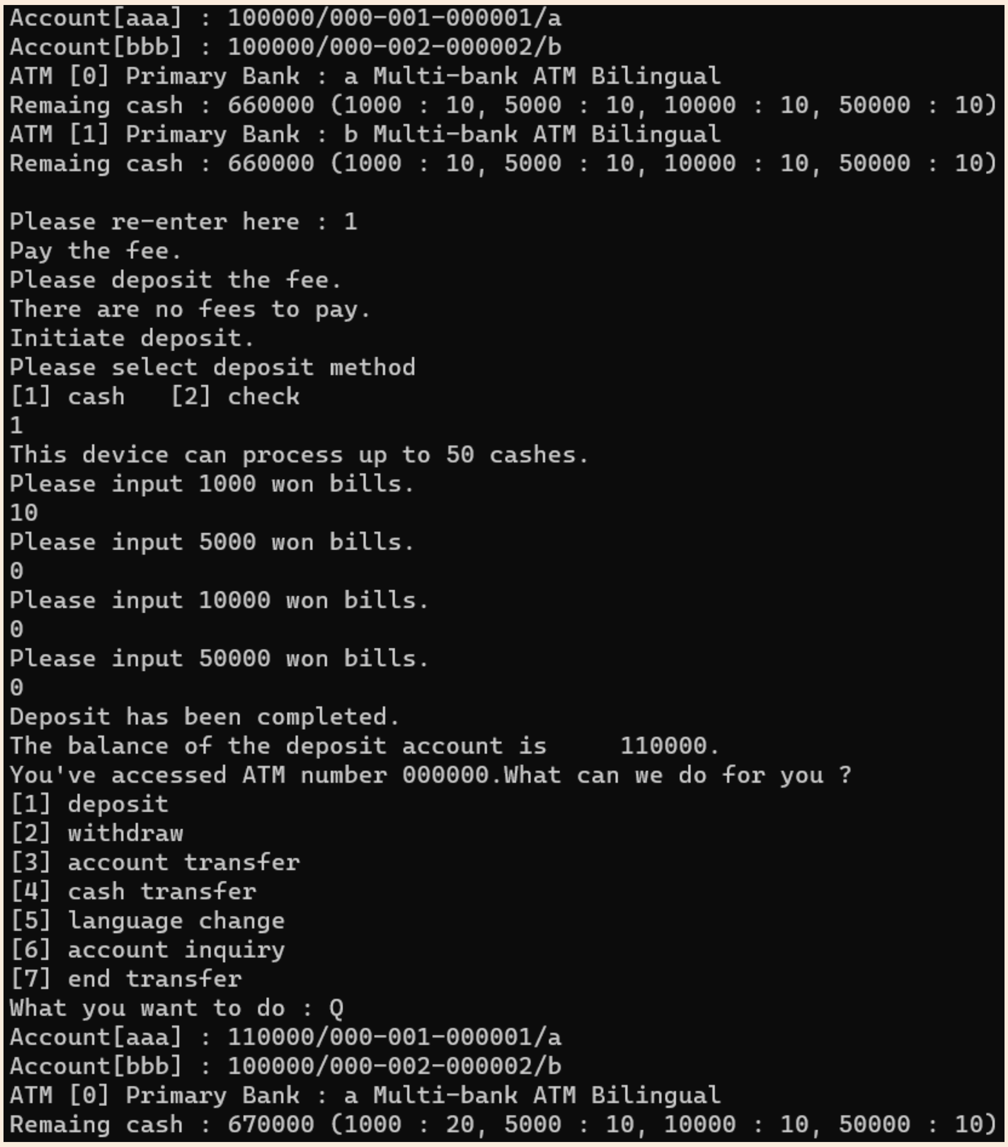
💡 입금에 수수료가 존재할 경우, 입금 수수료는 현금으로 받는다. 아래 이미지는, Multi Bank ATM에서 다른 은행의 계좌로 입금할 때, (1000원의) 수수료를 받는 이미지이다. 우리가 구현한 프로그램에서는 입금을 진행하기 전에 수수료를 먼저 받으며, 사용자들은 즉각적으로 보유한 현금의 일부를 수수료로 입금한다고 가정하였다.



```
What you want to do : 1
Deposits will be made to accounts at other banks.
Pay the fee.
Please deposit the fee.
The fee has been deposited.
Initiate deposit.
Please select deposit method
[1] cash   [2] check
```

- (REQ 4.5) The deposited cash increase available cash in ATM that can be used by other users.

💡 ATM에 현금을 입금하면, 입력된 현금은 ATM의 저장소로 들어가 다른 사용자들이 사용할 수 있게 된다. 50000원 1장, 1000원 1장을 입력하였을 경우, ATM에도 같은 지폐가 입력됨을 확인할 수 있다.

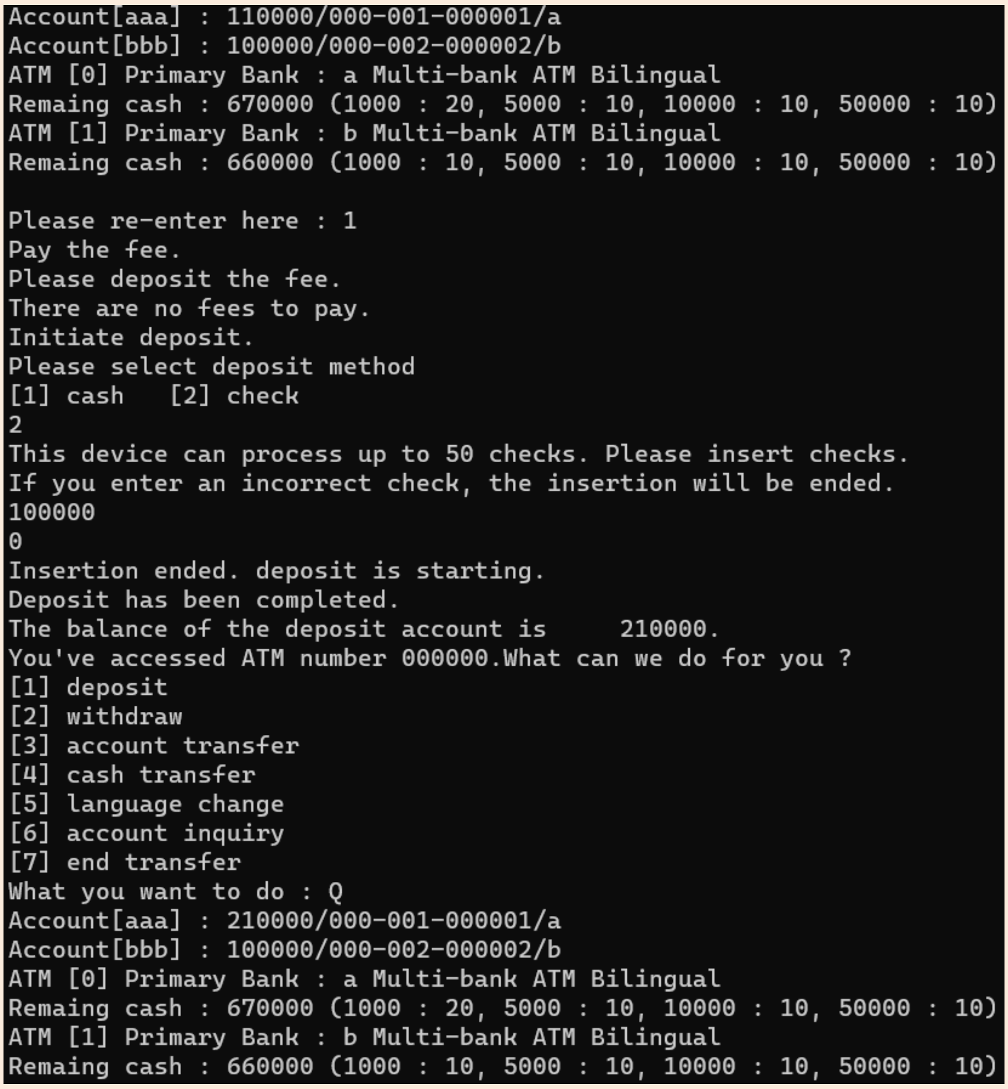


```
Account[aaa] : 100000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 1
Pay the fee.
Please deposit the fee.
There are no fees to pay.
Initiate deposit.
Please select deposit method
[1] cash    [2] check
1
This device can process up to 50 cashes.
Please input 1000 won bills.
10
Please input 5000 won bills.
0
Please input 10000 won bills.
0
Please input 50000 won bills.
0
Deposit has been completed.
The balance of the deposit account is     110000.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 110000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 670000 (1000 : 20, 5000 : 10, 10000 : 10, 50000 : 10)
```

- (REQ 4.6) The deposited check does not increase available cash in ATM that can be used by other users.

수표를 입력해도, ATM의 현금 저장소에 저장된 현금은 변하지 않는다.

아래 이미지는 계좌에 10만원의 수표 1장을 추가로 입금했을 때의 결과로, 거래를 진행하고 나서도 ATM의 현금 저장소에는 변동이 없음을 확인할 수 있다.

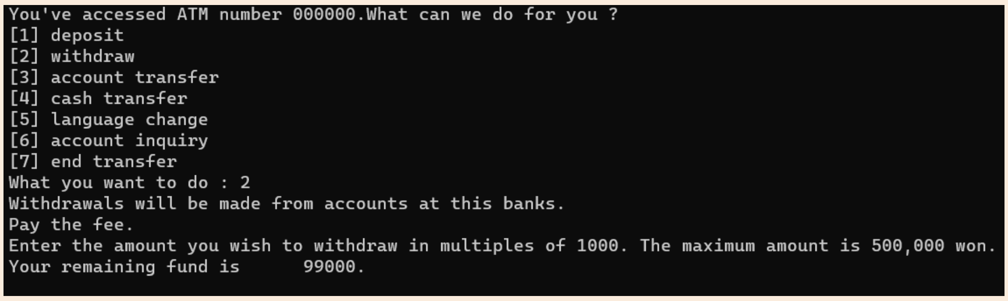


```
Account[aaa] : 110000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 670000 (1000 : 20, 5000 : 10, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 1
Pay the fee.
Please deposit the fee.
There are no fees to pay.
Initiate deposit.
Please select deposit method
[1] cash   [2] check
2
This device can process up to 50 checks. Please insert checks.
If you enter an incorrect check, the insertion will be ended.
100000
0
Insertion ended. deposit is starting.
Deposit has been completed.
The balance of the deposit account is     210000.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 210000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 670000 (1000 : 20, 5000 : 10, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

E. Withdrawal

- (REQ 5.1) An ATM shall ask a user to enter the amount of fund to withdraw.

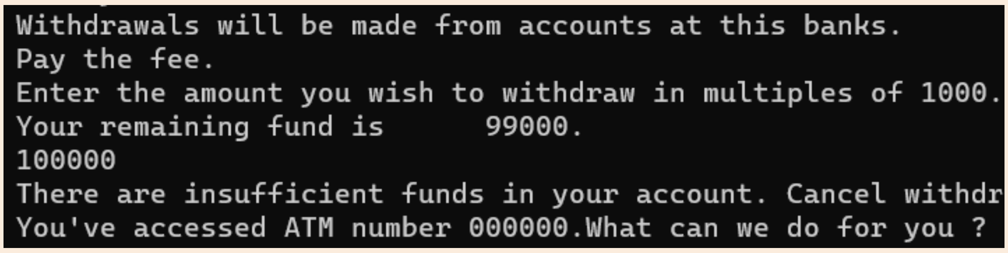
💡 사용자가 출금을 선택하면, ATM은 계좌의 잔액을 출력함과 동시에 출금할 액수를 묻는다.



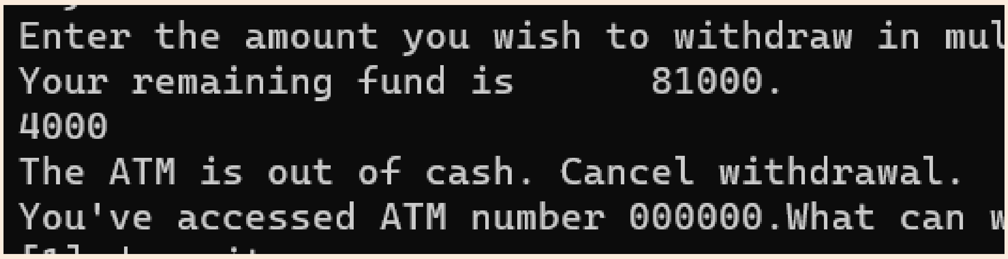
```
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 2
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is      99000.
```

- (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.

💡 계좌에 액수가 부족하거나, ATM에 현금이 부족하면 각각 적절한 오류 메시지를 출력하고 출금을 중단한다. 아래의 이미지는 각각 계좌에 액수가 부족한 상황에서 출금했을 경우, 그리고 ATM에 적절한 현금이 부족한 상황에서 출금했을 경우의 오류 메시지이다(예시에서는 ATM에 1000원 지폐가 부족한 상황에서 4000원을 출금하였다).



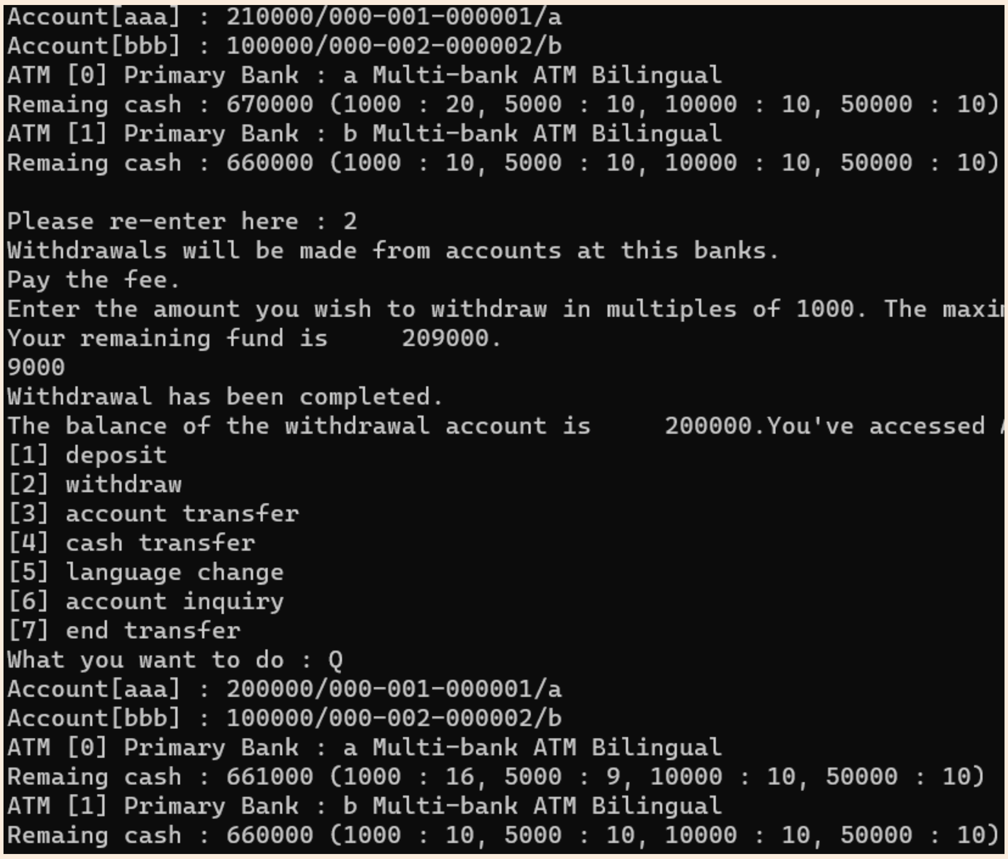
```
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000.
Your remaining fund is      99000.
100000
There are insufficient funds in your account. Cancel withdr
You've accessed ATM number 000000.What can we do for you ?
```



```
Enter the amount you wish to withdraw in mul
Your remaining fund is      81000.
4000
The ATM is out of cash. Cancel withdrawal.
You've accessed ATM number 000000.What can w
```

- (REQ 5.3) 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).

💡 출금이 완료되면, 계좌에서 출금한 양(+수수료)만큼의 금액이 빠져나간다. 다음 이미지는 출금 이전과 이후를 비교한 이미지로, 수수료 1000원과 출금액 9000원을 합해 총 10000원이 잔고에서 빠져나갔음을 확인할 수 있다.

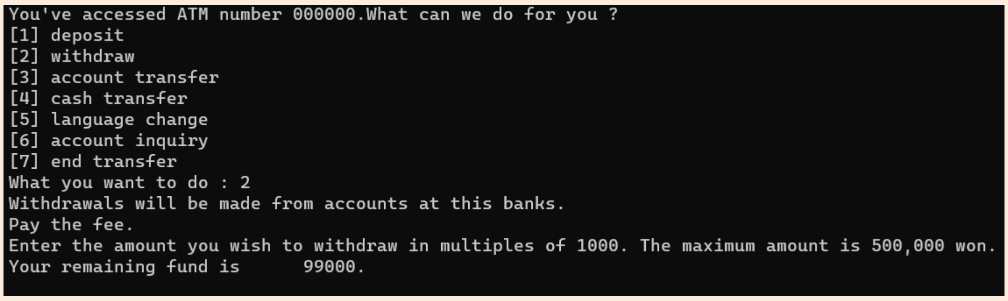


```
Account[aaa] : 210000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 670000 (1000 : 20, 5000 : 10, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 2
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maxi
Your remaining fund is     209000.
9000
Withdrawal has been completed.
The balance of the withdrawal account is     200000.You've accessed
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 661000 (1000 : 16, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

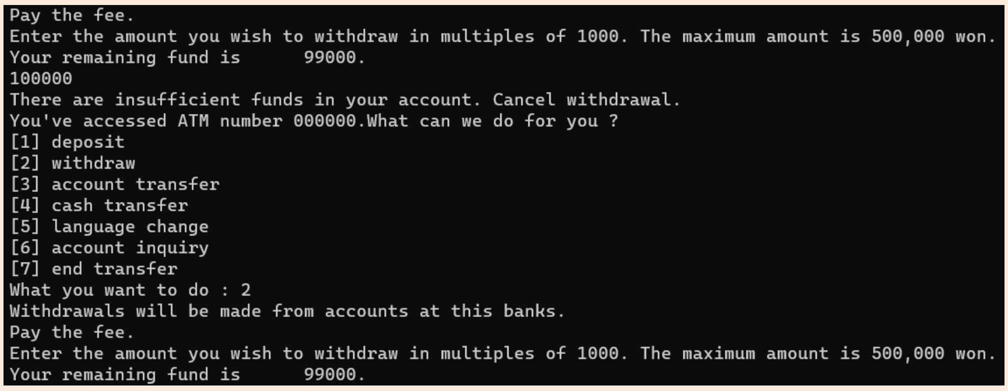
- (REQ 5.4) Some withdrawal fee may be charged (See REQ in System Setup).

💡 출금을 진행하기 이전, 계좌에 잔액이 충분할 경우, 출금 계좌에서 우선적으로 수수료를 차감한다. 이후, 출금이 중간에 중단되었을 경우, 차감한 수수료 만큼의 금액을 계좌로 다시 입금한다.

아래 이미지는 각각, 출금 시 수수료를 지불하는 이미지와 출금이 중단되었을 시 수수료 지불을 취소하는 이미지이다. 첫 번째 이미지에서, 계좌의 초기 잔고는 100000원이었으나, 1000원을 수수료로 지불하여 99000원이 되었다. 또한 두 번째 이미지에서, 첫 출금이 (잔액 이상을 입력하여) 실패하였을 경우, 계좌에 수수료가 반환되어 다음 출금 시에도 잔고가 유지됨을 확인할 수 있다.



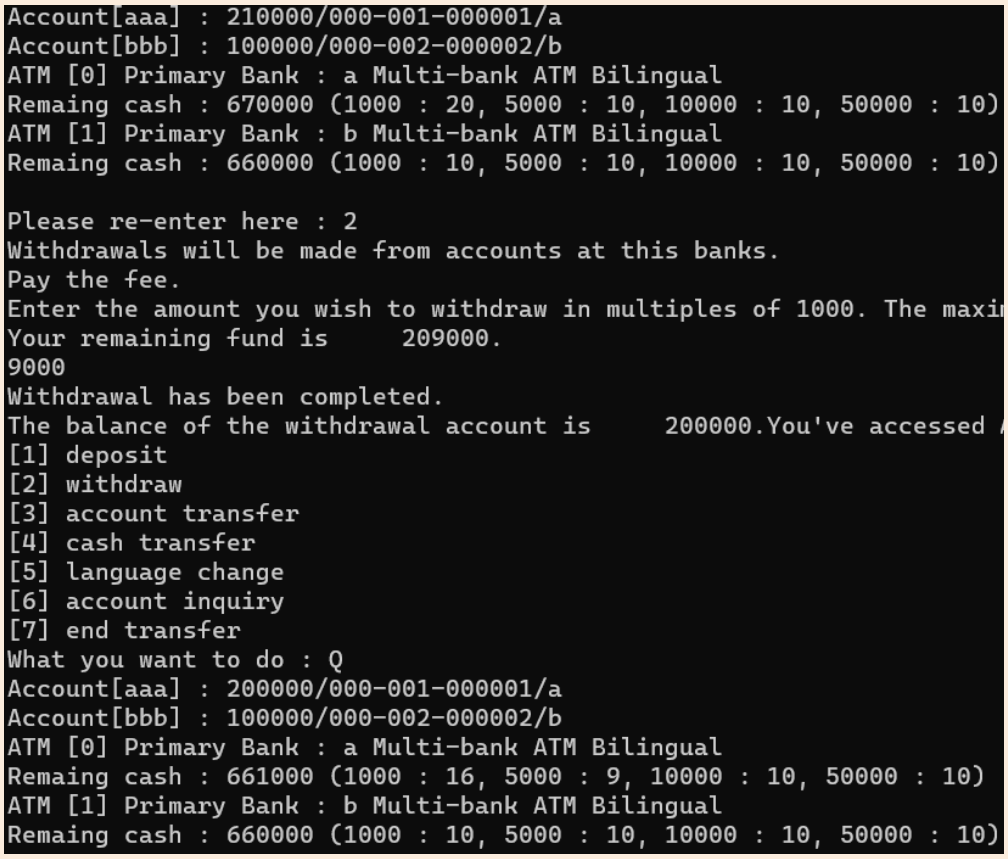
```
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 2
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is      99000.
```



```
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is      99000.
100000
There are insufficient funds in your account. Cancel withdrawal.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 2
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is      99000.
```

- (REQ 5.5) The cash withdrawal lower available cash in the ATM that can be used by other users.

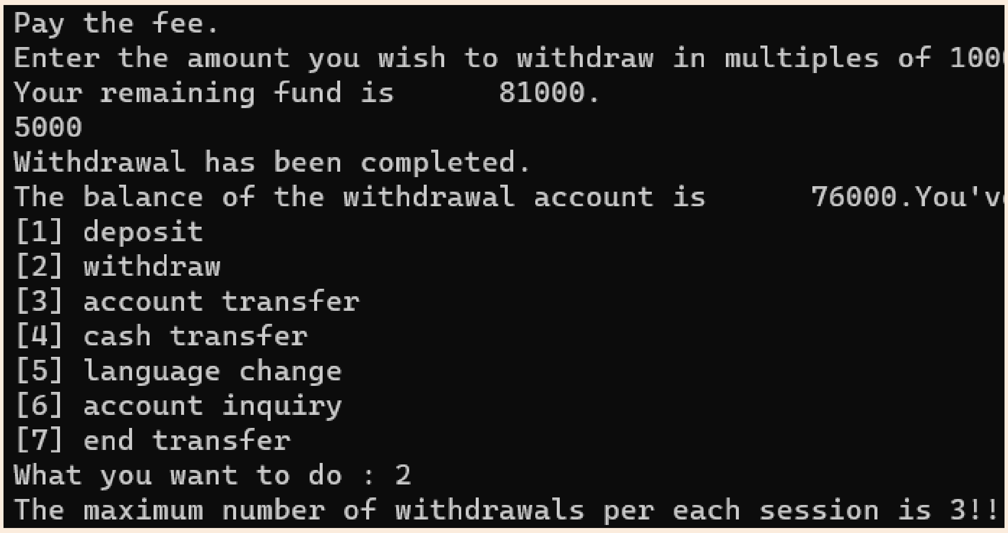
💡 출금은 ATM에 저장된 지폐의 수를 감소시킨다. 다음 이미지는 출금 이전과 이후를 비교한 이미지로, 1000원 4장(4000원)과 5000원 1장(5000원)이 저장소에서 빠져나갔음을 확인할 수 있다.



```
Account[aaa] : 210000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 670000 (1000 : 20, 5000 : 10, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 2
Withdrawals will be made from accounts at this banks.
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maxi
Your remaining fund is     209000.
9000
Withdrawal has been completed.
The balance of the withdrawal account is     200000.You've accessed
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 661000 (1000 : 16, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

- (REQ 5.6) The maximum number of withdrawals per each session is 3.

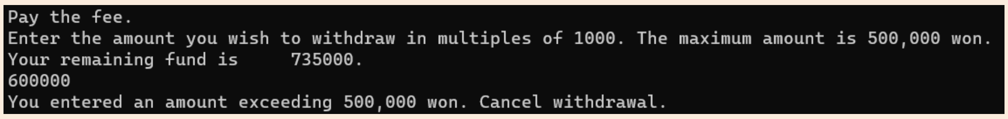
💡 세션 중 출금 횟수가 3회를 초과하면, 아래 이미지와 같은 오류 메시지를 출력하고 거래 선택 화면으로 되돌아간다.



```
Pay the fee.
Enter the amount you wish to withdraw in multiples of 100
Your remaining fund is      81000.
5000
Withdrawal has been completed.
The balance of the withdrawal account is      76000.You'v
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 2
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.

💡 한번에 500000원을 초과하는 값을 출금하려 시도할 시, 잔고가 충분하더라도 아래와 같은 오류 메시지와 함께 출금이 취소된다.

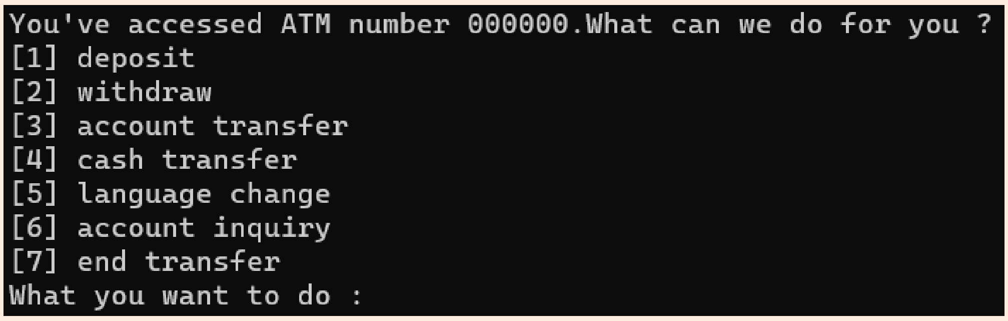


```
Pay the fee.
Enter the amount you wish to withdraw in multiples of 1000. The maximum amount is 500,000 won.
Your remaining fund is     735000.
600000
You entered an amount exceeding 500,000 won. Cancel withdrawal.
```

F. Transfer

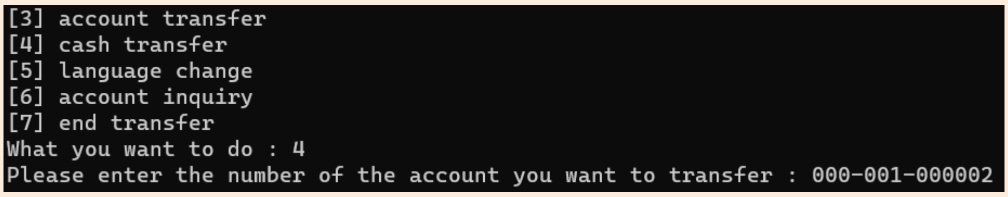
- (REQ 6.1) An ATM shall ask a user to choose the transfer types either cash transfer or account fund transfer.

💡 우리의 구현에서는 계좌 송금과 현금 송금을 완전히 다른 종류의 거래로 취급하여, 거래 종류 선택 단계에서 다음과 같이 분리하였다.


```
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do :
```

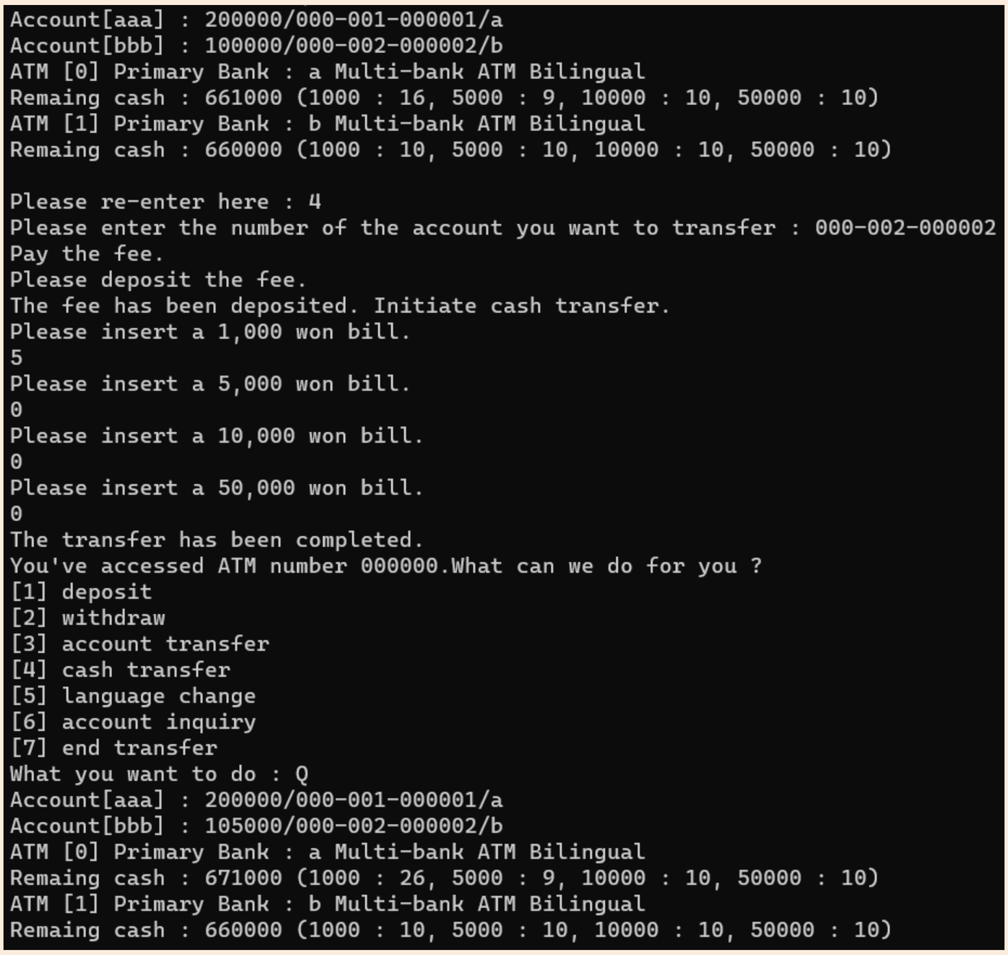
- (REQ 6.2) For both cash and account transfers, an ATM shall ask the destination account number where the fund is to be transferred.

💡 송금 시, 송금을 받을 계좌의 번호를 다음 예시와 같이 입력받는다.


```
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 4
Please enter the number of the account you want to transfer : 000-001-000002
```

- (REQ 6.3) For cash transfer, an ATM shall ask user to insert the cash including the transaction fees, and verify if the amount of the inserted cash is correct. All inserted cash excluding the transaction fee shall be transferred.

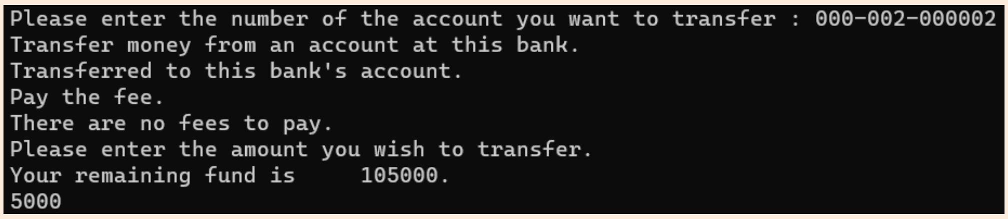
💡 현금 송금 시, 아래와 같이 수수료와 송금 금액을 별도로 받는다. 수수료는 입금 시와 동일하게, 사용자가 보유한 현금으로 즉각적으로 납부한다고 가정한다. 아래의 이미지는 현금 송금 과정에서의 수수료와 송금 금액의 납부, 현금 송금 전후의 계좌 잔액 비교이다. 대상 계좌에 총 5000원이 송금되었음을 확인할 수 있다.



```
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 661000 (1000 : 16, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 4
Please enter the number of the account you want to transfer : 000-002-000002
Pay the fee.
Please deposit the fee.
The fee has been deposited. Initiate cash transfer.
Please insert a 1,000 won bill.
5
Please insert a 5,000 won bill.
0
Please insert a 10,000 won bill.
0
Please insert a 50,000 won bill.
0
The transfer has been completed.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 105000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 671000 (1000 : 26, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

- (REQ 6.4) For account transfer, an ATM shall ask the source account number, and the amount of fund to be transferred.

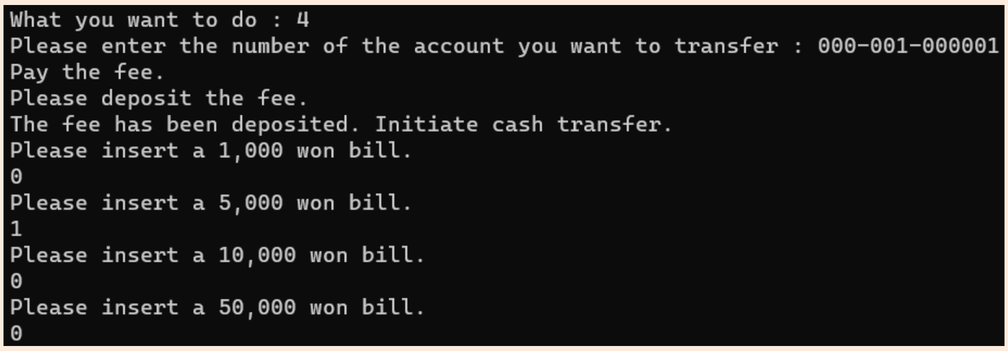
💡 우리의 구현에서, card는 모두 account와 연결되어 있는 체크카드이다. 따라서, ATM은 자동으로 source account를 입력받으므로, 아래와 같이 destination account 및 fund만 수동으로 입력받는다.



```
Please enter the number of the account you want to transfer : 000-002-000002
Transfer money from an account at this bank.
Transferred to this bank's account.
Pay the fee.
There are no fees to pay.
Please enter the amount you wish to transfer.
Your remaining fund is     105000.
5000
```

- (REQ 6.5) Some transfer fee may be charged (See REQ in System Setup).

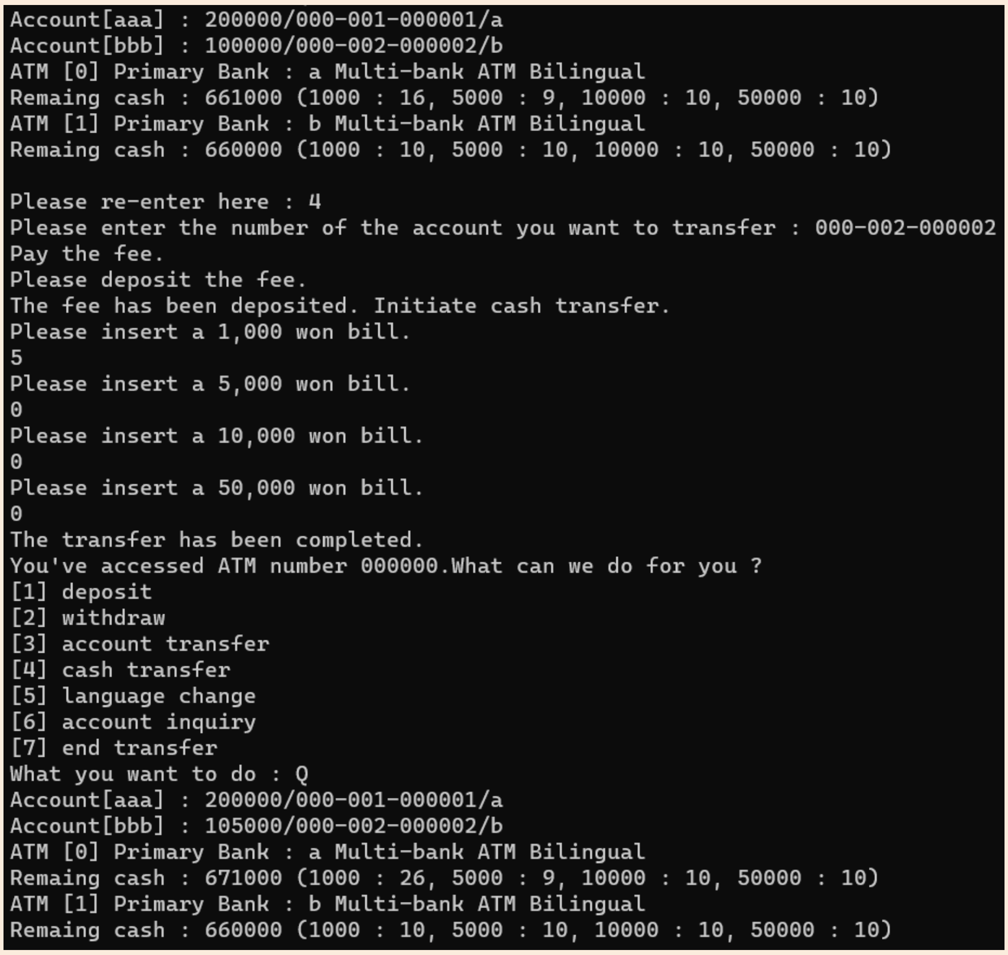
💡 송금 이전, 아래와 같이 수수료를 현금으로 받거나, 계좌에서 자동으로 차감한다. 계좌 송금이 실패할 경우, 차감된 수수료는 계좌로 반환된다.



```
What you want to do : 4
Please enter the number of the account you want to transfer : 000-001-000001
Pay the fee.
Please deposit the fee.
The fee has been deposited. Initiate cash transfer.
Please insert a 1,000 won bill.
0
Please insert a 5,000 won bill.
1
Please insert a 10,000 won bill.
0
Please insert a 50,000 won bill.
0
```

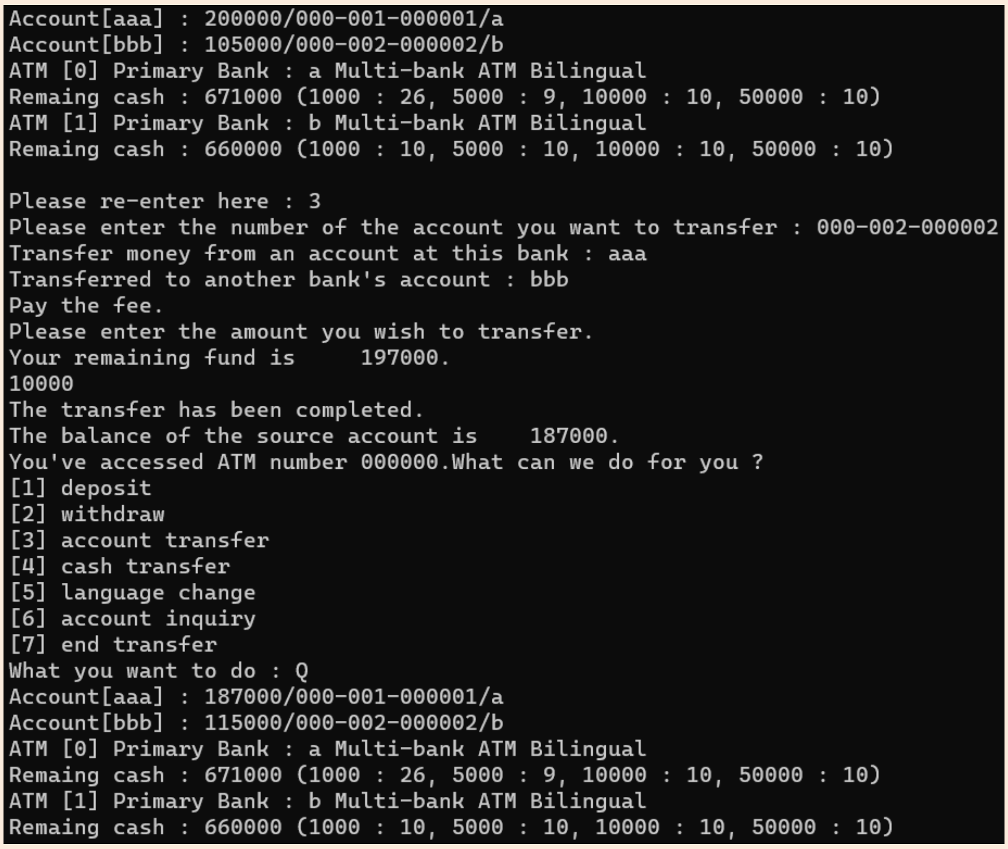
- (REQ 6.6) The inserted cash for transfer increase available cash in ATM that can be used by other users.

💡 현금 송금 후, 수수료와 투입된 금액은 ATM에 저장된다. 다음 이미지는 현금 송금 이전과 이후를 비교한 결과이다. 우리는 수수료를 수동으로 설정할 수 있도록 구현하였으며, 예시와 달리 수동으로 설정된 수수료가 5000원이 아닐 수 있는 관계로, 1000원권으로 받도록 설정하였다. 따라서, 수수료 5000원(1000원 * 5장)과 투입 금액(1000원 * 5장)을 합해 총 1000원권 10장이 ATM에 저장되었음을 확인할 수 있다.


```
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 100000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 661000 (1000 : 16, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 4
Please enter the number of the account you want to transfer : 000-002-000002
Pay the fee.
Please deposit the fee.
The fee has been deposited. Initiate cash transfer.
Please insert a 1,000 won bill.
5
Please insert a 5,000 won bill.
0
Please insert a 10,000 won bill.
0
Please insert a 50,000 won bill.
0
The transfer has been completed.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 105000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 671000 (1000 : 26, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

- (REQ 6.7) 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).

💡 송금 후, 송금된 금액은 쌍방의 계좌에 반영되고, 계좌 송금의 경우 송금 계좌에서 수수료도 빠져나간다. 다음 이미지는 계좌 송금 이전과 이후를 비교한 이미지로, aaa의 계좌에서는 수수료 3000원 + 송금액 10000원을 합한 13000원이 빠져나가고, bbb의 계좌에 송금액 10000원이 입금되었음을 확인할 수 있다.

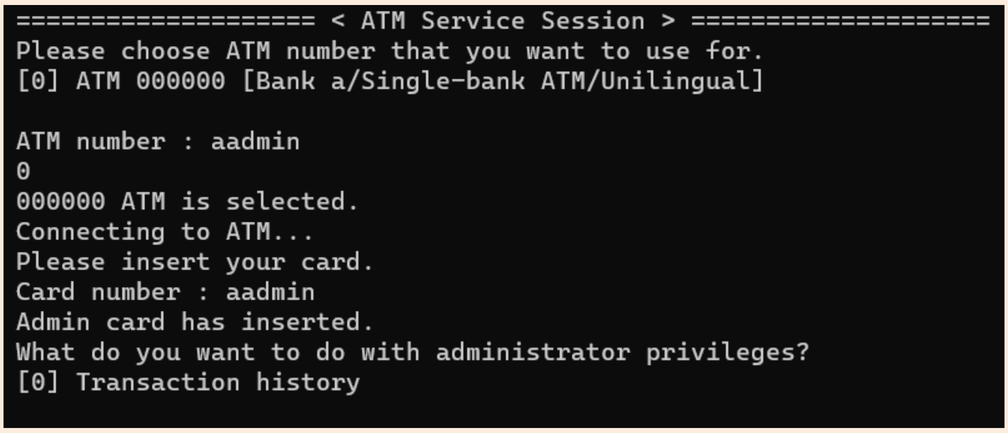


```
Account[aaa] : 200000/000-001-000001/a
Account[bbb] : 105000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 671000 (1000 : 26, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

Please re-enter here : 3
Please enter the number of the account you want to transfer : 000-002-000002
Transfer money from an account at this bank : aaa
Transferred to another bank's account : bbb
Pay the fee.
Please enter the amount you wish to transfer.
Your remaining fund is     197000.
10000
The transfer has been completed.
The balance of the source account is    187000.
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : Q
Account[aaa] : 187000/000-001-000001/a
Account[bbb] : 115000/000-002-000002/b
ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 671000 (1000 : 26, 5000 : 9, 10000 : 10, 50000 : 10)
ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)
```

G. Display of Transaction History (Admin Menu)

- (REQ 7.1) 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.

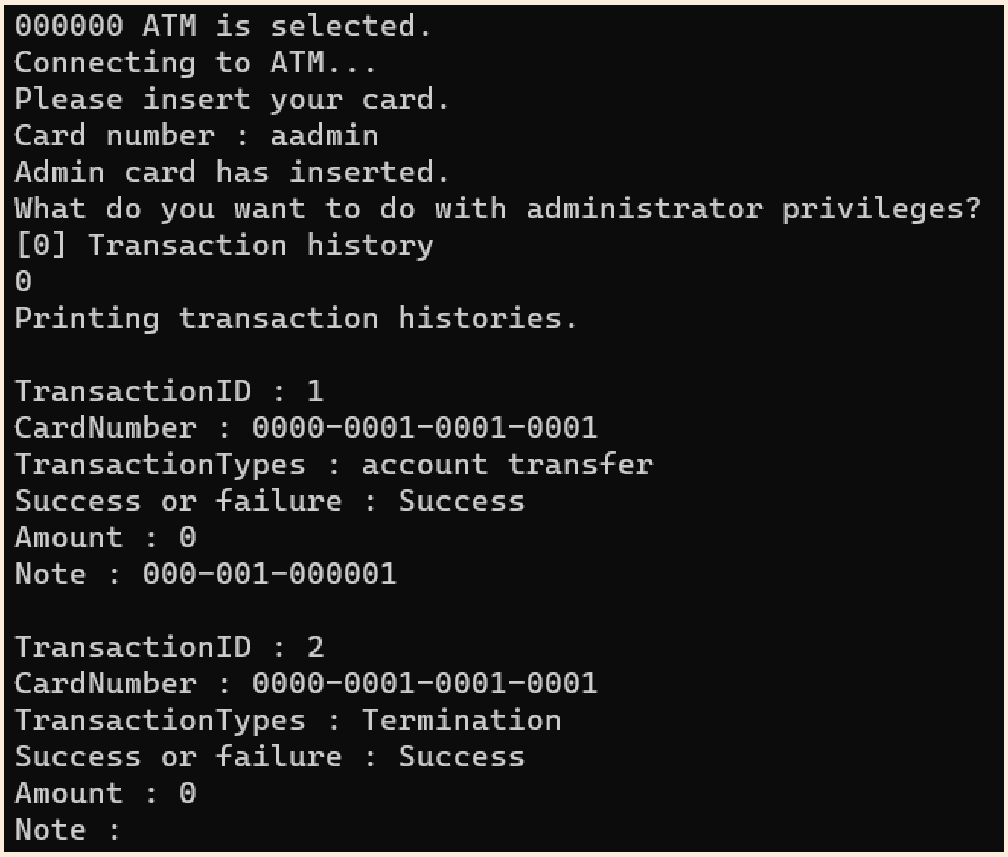
💡 세션을 시작할 때, 카드 대신 admin card key(ATM을 생성할 때 자동으로 생성된다)를 삽입하면, 다음과 같이 관리자 세션에 진입할 수 있다. 이때, ATM은 “거래 내역 보기” 메뉴만을 출력한다.



```
==================== < ATM Service Session > ====================
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank a/Single-bank ATM/Unilingual]

ATM number : aadmin
0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : aadmin
Admin card has inserted.
What do you want to do with administrator privileges?
[0] Transaction history
```

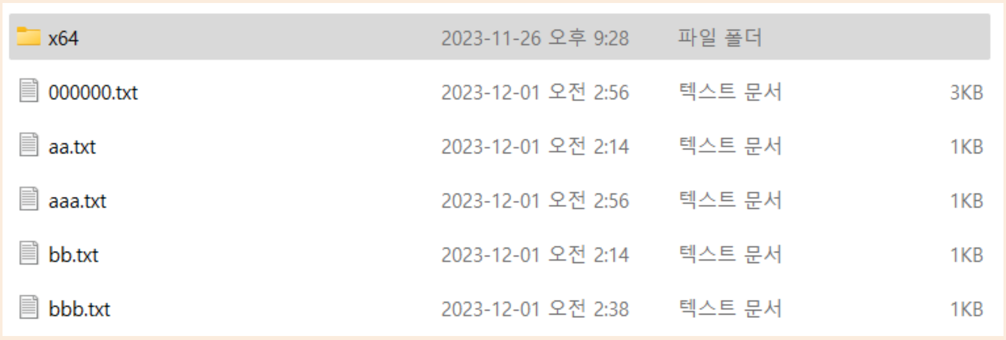
- (REQ 7.2) When the “Transaction History” menu is selected, an ATM display the information of all transactions from all users from the beginnig of the system start.

💡 거래 내역 보기 기능을 선택하면, 시스템이 실행된 후부터 진행된 모든 거래가 다음과 같이 출력된다.


```
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : aadmin
Admin card has inserted.
What do you want to do with administrator privileges?
[0] Transaction history
0
Printing transaction histories.

TransactionID : 1
CardNumber : 0000-0001-0001-0001
TransactionTypes : account transfer
Success or failure : Success
Amount : 0
Note : 000-001-000001

TransactionID : 2
CardNumber : 0000-0001-0001-0001
TransactionTypes : Termination
Success or failure : Success
Amount : 0
Note :
```

- (REQ 7.3) The “Transaction History” information shall be outputted to the external file.

💡 ATM의 거래 내역은 다음 이미지에서 확인할 수 있듯, 프로젝트 폴더 내에 ATM 일련번호와 동일한 이름을 가진 텍스트 파일에 저장된다.

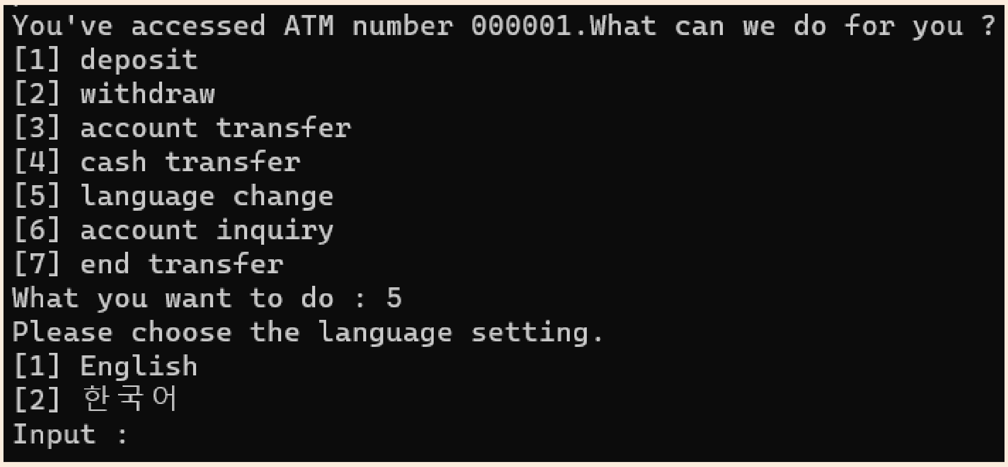


x64	2023-11-26 오후 9:28	파일 폴더	
000000.txt	2023-12-01 오전 2:56	텍스트 문서	3KB
aa.txt	2023-12-01 오전 2:14	텍스트 문서	1KB
aaa.txt	2023-12-01 오전 2:56	텍스트 문서	1KB
bb.txt	2023-12-01 오전 2:14	텍스트 문서	1KB
bbb.txt	2023-12-01 오전 2:38	텍스트 문서	1KB

H. Multi-language Support

- (REQ 8.1) 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.

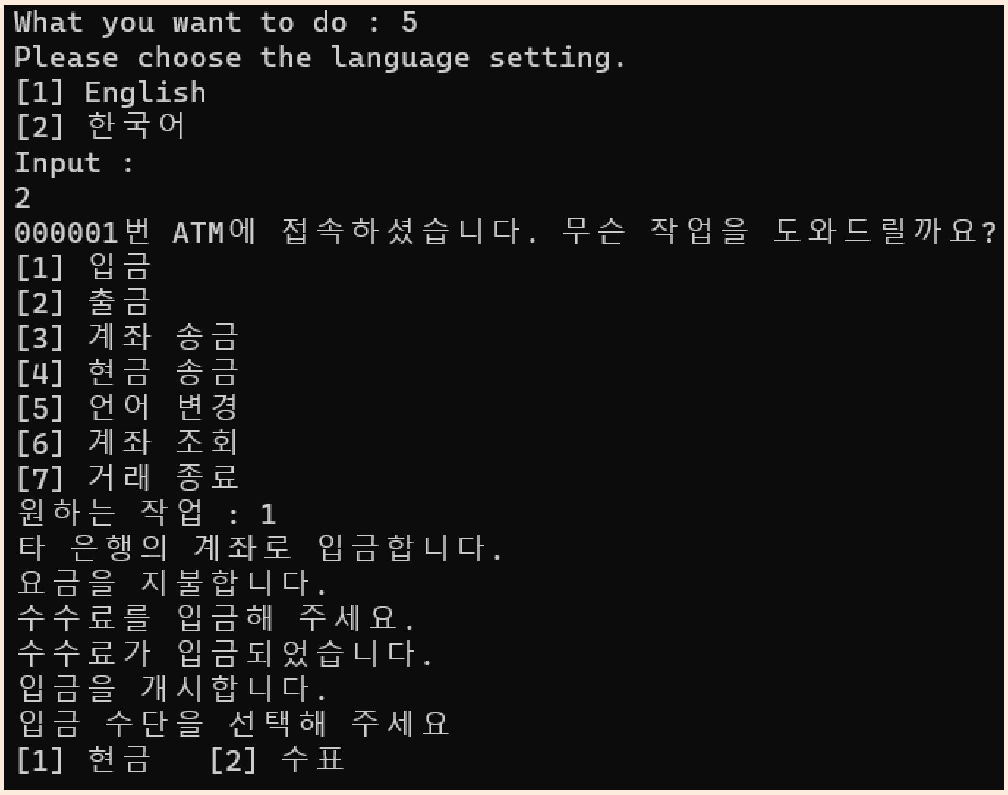
> 💡 Bilingual ATM에서 language change 기능을 선택하면, 다음과 같이 언어를 선택할 수 있는 창이 출력된다. 이때 현재의 언어 설정과 무관하게, 각 선택지는 해당하는 언어로 주어진다(e.g. [1] English, [2] 한국어, [3] 日本語…).



```
You've accessed ATM number 000001.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do : 5
Please choose the language setting.
[1] English
[2] 한국어
Input :
```

- (REQ 8.2) Once a certain language is chosen, all menus must be displayed using the chosen language.

💡 언어 설정을 완료하면, 다음과 같이 다시 작업 선택 화면으로 돌아와 모든 선택지를 설정된 언어로 표시한다.

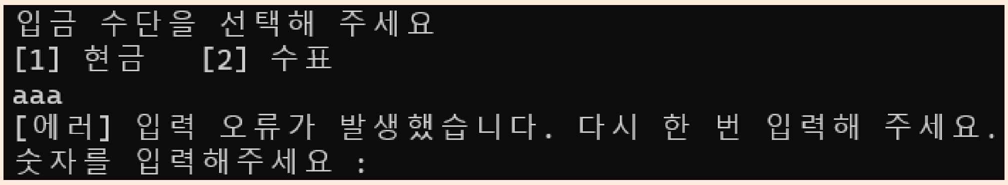


```
What you want to do : 5
Please choose the language setting.
[1] English
[2] 한국어
Input :
2
000001번 ATM에 접속하셨습니다. 무슨 작업을 도와드릴까요?
[1] 입금
[2] 출금
[3] 계좌 송금
[4] 현금 송금
[5] 언어 변경
[6] 계좌 조회
[7] 거래 종료
원하는 작업 : 1
타 은행의 계좌로 입금합니다.
요금을 지불합니다.
수수료를 입금해 주세요.
수수료가 입금되었습니다.
입금을 개시합니다.
입금 수단을 선택해 주세요
[1] 현금   [2] 수표
```

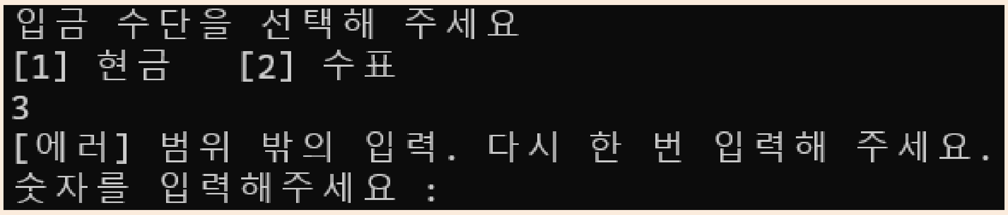
I. Exception Handling

- (REQ 9.1) An ATM shall display an appropriate message for each exception scenario (both explicitly stated in this document and implicitly assumed ones), and take an appropriate action.

💡 적절하지 않은 입력이 입력된 경우, ATM은 다음과 같이 오류 메시지를 출력한다. 이때 같은 자료형이되 주어진 선택지의 범위를 벗어나는 경우에는 다음과 같이 "범위 밖의 입력" 메시지를 출력하고, 다른 자료형이 입력되었을 경우에는 "입력 오류 발생" 메시지를 출력한다. 그 이외에, 각 요구조건에서 상정된 예외의 경우에는 해당 요구조건 항목에 예외 처리 방식이 각각 기술되어 있다.



```
입금 수단을 선택해 주세요
[1] 현금   [2] 수표
aaa
[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요.
숫자를 입력해주세요 :
```

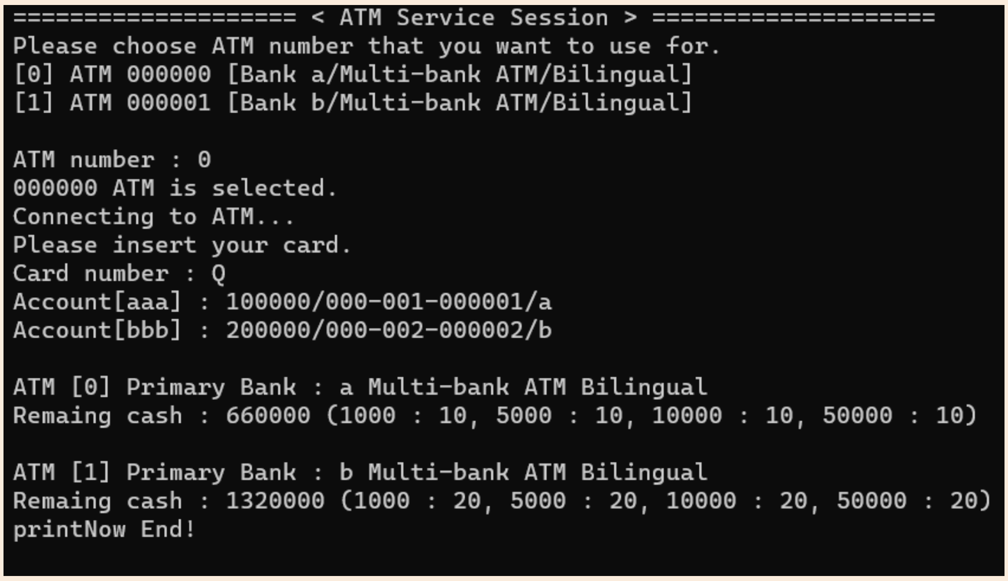


```
입금 수단을 선택해 주세요
[1] 현금   [2] 수표
3
[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요.
숫자를 입력해주세요 :
```

J. Display of Account/ATM Snapshot

- (REQ 10.1) When a particular character is given as a console input during the program execution, the following information shall be displayed to the console.

💡 아무 입력창에서 Q를 입력하면, 현존하는 모든 account의 정보(잔고, 계좌 번호 및 은행 정보)가 출력된 후, 현존하는 모든 ATM의 정보(은행, 타입, 언어, 잔고)가 출력된다.



```
==================== < ATM Service Session > ====================
Please choose ATM number that you want to use for.
[0] ATM 000000 [Bank a/Multi-bank ATM/Bilingual]
[1] ATM 000001 [Bank b/Multi-bank ATM/Bilingual]

ATM number : 0
000000 ATM is selected.
Connecting to ATM...
Please insert your card.
Card number : Q
Account[aaa] : 100000/000-001-000001/a
Account[bbb] : 200000/000-002-000002/b

ATM [0] Primary Bank : a Multi-bank ATM Bilingual
Remaing cash : 660000 (1000 : 10, 5000 : 10, 10000 : 10, 50000 : 10)

ATM [1] Primary Bank : b Multi-bank ATM Bilingual
Remaing cash : 1320000 (1000 : 20, 5000 : 20, 10000 : 20, 50000 : 20)
printNow End!
```

3. Concepts of Object-Oriented Programming

A. Object-Oriented Programming이란?

- Code를 어떻게 짤지와 같은 Programming 방법론에는 여러가지가 있는데, 옛날에는 필요할 때 변수와 함수를 그때 그때 만들어 쓰는 Procedure Based Programming을 많이 했었다.
- 그런데 이런 Procedure Based Programming은 프로그램의 규모가 커지면 변수와 function의 양이 많아져서 변수와 해당 변수를 다룰 function을 맞춰서 연산하기가 어려워진다. 그래서 나오게 된 Programming 방법론(Paradigm)이 Object Oriented Programming이다.
- Object Oriented Programming은 Program이 Attribute와 Function을 가진 하나의 module을 단위로 돌아가도록 Program의 Code를 module 단위로 짜는 방법이다.

- 이러한 Object-Oriented Programming을 지원하는 컴퓨터 언어의 예시로는 C++, C#, Java, Python, R, PHP 등이 있다.
- 이러한 Object-Oriented Programming을 하는 이유로는 다음의 세가지를 뽑을 수 있다.
    - Robustness (강건성)
        - 어떤 Software를 만들기 위해서 Programming을 할 때, 우리는 만드는 Software가 unexpected application(예상못한 적용분야)에서 unexpected inputs(예상못한 입력들)에도 적절히 대응할 수 있는 것을 바란다.
        - 쉽게 말해서 우리는 만드는 Software가 튼튼하게 오래 유지 관리되는 것을 바란다는 것이다.
        - Software의 이런 특성을 Robustness (강건성)라고 하는데 Object Oriented Programming(OOP) 방법은 방법 자체가 Software에 크던 작던 Robustness (강건성)를 부여하며, Object Oriented Programming(OOP) 방법으로 Programming을 할 때에도 Robustness (강건성)을 부여하려는 목적의식을 가지고 Programming을 해야한다는 약속이 있다.
    - Adaptability (적절성)
        - 어떤 Software를 만들어서 쓸 때, 우리는 그 Software가 사용되는 환경 내에서 시간이 지남에 조건이 변함에도 원하는 연산을 정확하고 빠르게 처리할 수 있는 것을 바란다.
        - 쉽게 말해서 우리는 속성과 그 속성을 위한 method들을 잘 짝지어서 써서 data가 망가지지 않게 원하는 연산을 정확하고 빠르게 처리하는 것을 바란다는 것이다.
        - Software의 이런 특성을 Adaptability (적절성)라고 하는데 Object Oriented Programming(OOP) 방법은 방법 자체가 Software에 크던 작던 Adaptability (적절성)를 부여하며, Object Oriented Programming(OOP) 방법으로 Programming을 할 때에도 Adaptability (적절성)을 부여하려는 목적의식을 가지고 Programming을 해야한다는 약속이 있다.
    - Reusability (재활용성)
        - 어떤 Software를 만들어서 쓸 때, 만약 다른 system에서 우리가 쓰는 Software와 같은 연산을 처리하고자 할 때, 우리는 Software에 활용된 Code를 떼어다가 다른 system의 내부 코드로 활용할 수 있기를 바란다.
        - Software의 이런 특성을 Reusability (재활용성)라고 하는데 Object Oriented Programming(OOP) 방법은 방법 자체가 Software에 크던 작던 Reusability (재활용성)를 부여하며, Object Oriented Programming(OOP) 방법으로

Programming을 할 때에도 Reusability (재활용성)을 부여하려는 목적의식을 가지고 Programming을 해야한다는 약속이 있다.

B. Abstraction

- Abstraction이란 Object Oriented Programming(OOP)을 통해 module을 만들 때에는, 나중에 module을 사용하게 될 사용자가 module이 받은 인자들을 가지고 어떻게 연산해서 결과값을 만들어내는지를 자세하게 알고 있지 않아도 쉽게 module을 사용할 수 있도록 만들어야 한다는 Concept이다. 이렇게 만들게 되면 여러가지 Object들 사이에서 Attribute, Behavior, Association 등 그 관계를 추상화 할 수 있다.
- C++에서 이러한 Abstraction을 위해 제공하는 기능으로는 다음과 같은 것들이 있다.
    - Class Properties: Class를 통해 만들고자 하는 Data Type의 속성(variable, object properties)을 말하며, Class의 Definition 안에서 변수를 Declaration 및 Initialization한 것들을 말한다.
    - Class Behaviors: Class를 통해 만들어진 Data Type을 가지는 Instance들을 다루는 방법(method, available interactions with an object)들을 말하며, Class의 Definition 안에서 함수를 Declaration 및 Definition한 것들을 말한다.

C. Encapsulation

- Encapsulation이란 Object Oriented Programming(OOP)을 통해 module을 만들 때에는, 나중에 module을 사용하게 될 사용자가 module의 내부에 구축된 연관성을 module 외부에서 접근해서 깨뜨리지 못하도록 사용자가 module의 내부에 구축된 연관성에 접근하지 못하게 차단하는 방식으로 만들어야 한다는 Concept이다.
- C++에서 이러한 Encapsulation을 위해 제공하는 기능으로는 다음과 같은 것들이 있다.
    - Class Access Specifiers: C++에서는 Class를 만들 때, Encapsulation을 위해 Class 안에서 이 Specifier에 속한 모든 Class Member(Variable, Function 모두 해당)는 그 어떤 접근 방법으로도 접근이 가능하도록 설정하는 Public Specifier과 Class 안에서 이 Specifier에 속한 모든 Class Member(Variable, Function 모두 해당)는 Class 내부에서만 접근이 가능하도록 설정하는 Private Specifier, Class 안에서 이 Specifier에 속한 모든 Class Member(Variable, Function 모두 해당)는 이 Class를 상속을 받을 다른 Class가 있을 시 해당 상속을 받는 Class에서 항상 접근이 가능하도록 설정하는 Protected Specifier를 제공한다.

D. Polymorphism

- Polymorphism (다형성)이란 하나의 Interface로 서로 다른 Type들을 접근할 수 있도록 할 수 있도록 하는 Provision(규정) 혹은 하나의 Symbol을 서로 다른 여러 개의 Type들로서 사용할 수 있도록 만들어야 한다는 Concept이다.
- C++에서 이러한 Polymorphism을 위해 제공하는 기능으로는 다음과 같은 것들이 있다.
    - Function Overloading
        - 위와 같은 Polymorphism의 가장 대표적인 예시가 바로 Function Overloading이다.
        - 이러한 Function Overloading이란 Function의 이름은 같게 쓰되 Function의 Return Type이나 Input Parameter의 갯수나 Type을 서로 다르게 해서 여러 개 정의하는 것을 말한다. 그런데 여기서 주의할 점은 C++에서는 Function의 이름과 Input Parameter들은 동일한데 Return Type만 다르게 쓰는 것은 Function Overloading으로 인정하지 않는다는 것이다.

- sum : 1 + 3
- sum : 1 + 3.5
- sum : 1.5 + 3
- sum : 1 + 3 + 2
- sum : ‘a’ + ‘b’

```
int Sum ( int, int);
double Sum ( int, double);
double Sum (double , int);
int Sum ( int, int, int);
char* Sum (char, char);
```

-
    - Operator Overloading
        - 위와 같은 Polymorphism의 또 다른 대표적인 예시가 바로 Operator Overloading이다.
        - C++에서 Operator들은 모두 그 Operator에 대응하는 Keyword Function들이 있고 해당 Keyword Function들을 Overloading하는 것으로 해당 Operator를 Overloading할 수 있다.

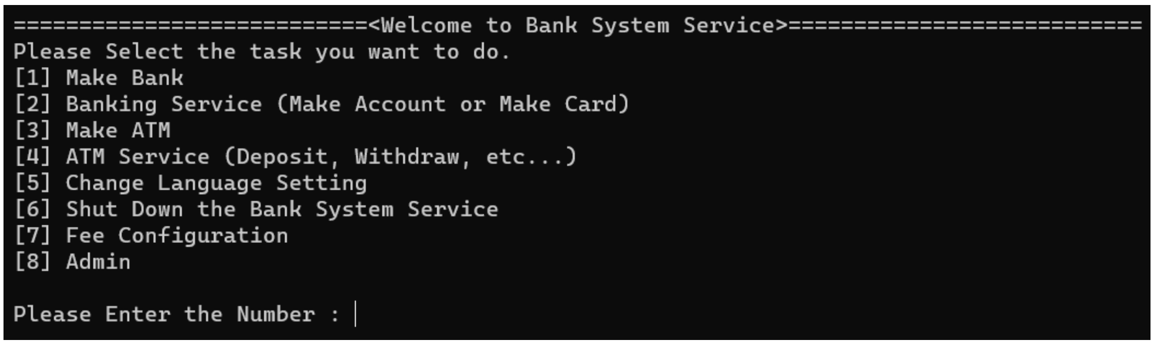
E. Inheritance

- Object-Oriented Programming(OOP)에서 Class간의 계층 구조를 만들어주기 위해 사용하는 개념으로 Inheritance (상속)이라는 개념이 있다.

- Inheritance (상속)은 Base가 되는 Class가 있고, 새로운 SubClass를 만들 때, 그 SubClass가 상위 Class를 상속받아 상위 Class의 Member들을 상속받도록 하는 것을 말한다. 상위 Class를 Specializing하기 위해 사용한다고 볼 수도 있고, 하위 Class를 Extending하기 위해 사용한다고 볼 수도 있다. 따라서 이 Inheritance (상속)은 Class 간의 계층 구조를 만드는 방식으로 동작한다.
- 이렇게 Inheritance (상속)을 통해 Class간의 계층 구조를 만들어 두면 BaseClass를 다루는 것만으로도 BaseClass의 하위의 모든 SubClass를 한꺼번에 다룰 수 있다는 특징이 있다. 이 특징은 구현된 Data 구조나 프로그램의 유지 및 보수에 도움이 되기 때문에 장점이 된다.
- C++에서 이러한 Inheritance을 위해 제공하는 기능으로는 다음과 같은 것들이 있다.
    - Inheritance Specifiers: C++에서는 Base Class에서 Public 및 Protected 속성을 가지는 Member들은 Derived Class에서도 그 속성을 유지하되 Base Class에서 Private 속성을 가지는 Member들은 Derived Class에서 그 어떠한 경우에도 Inaccessible하게 만드는 public Inheritance Specifier를 제공하고 있으며, Base Class에서 Public 및 Protected 속성을 가지는 Member들은 Derived Class에서 Protected 속성을 유지하되 Base Class에서 Private 속성을 가지는 Member들은 Derived Class에서 그 어떠한 경우에도 Inaccessible하게 만드는 protected Inheritance Specifier를 제공하고 있다. 또한, Base Class에서 Public 및 Protected 속성을 가지는 Member들은 물론이고 Base Class에서 Private 속성을 가지는 Member들을 모두 Derived Class에서 그 어떠한 경우에도 Inaccessible하게 만드는 Private Inheritance Specifier를 제공하고 있다.
    - Function Overriding: Function Overriding이란 Base Class에서도 정의된 Function이 Derived Class에서 Base Class에서 하는 작업보다 추가적인 작업 (added job)을 해주기를 바란다거나, Base Class와는 다른 작업(Different job)을 하기를 바랄 때 사용하는 Programming 방법이다. 이러한 Function Overriding은 어떤 한 Class가 다른 Class로부터 Inherited(상속받은) Derived Class인 상황에서만 사용할 수 있는 Programming 방법이다. 이러한 Function Overriding의 Function Overloading과 비교되는 강력한 특징은 Derived Class에서 Base Class의 Member Function을 Signature를 완전히 동일하게 하여 Redefine한다는 것이다.

4. Instruction to run the source code

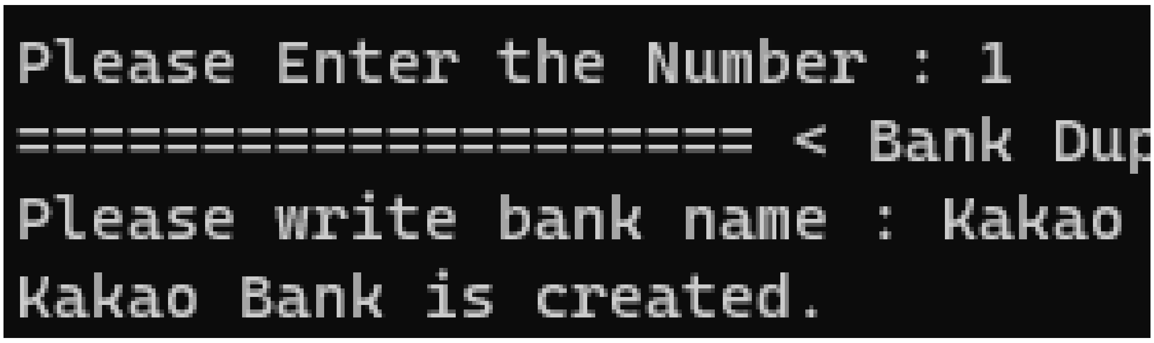
- main.cpp를 실행하면 세션이 시작된다.



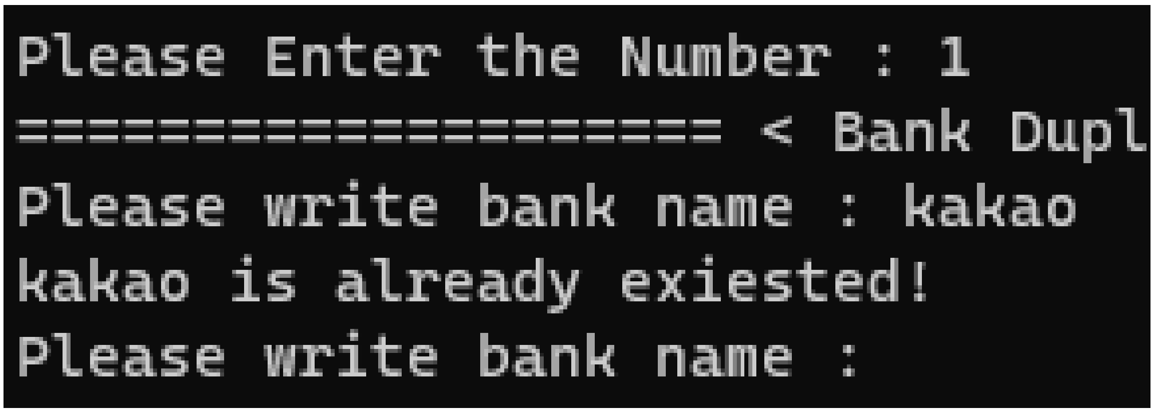
```
==========================<Welcome to Bank System Service>==========================
Please Select the task you want to do.
[1] Make Bank
[2] Banking Service (Make Account or Make Card)
[3] Make ATM
[4] ATM Service (Deposit, Withdraw, etc...)
[5] Change Language Setting
[6] Shut Down the Bank System Service
[7] Fee Configuration
[8] Admin

Please Enter the Number :
```

그러면 위와 같은 기능과 번호가 나타난다. 원하는 기능의 번호를 콘솔에 입력하고 엔터를 치면 해당 기능이 시작된다.

1. Make Bank



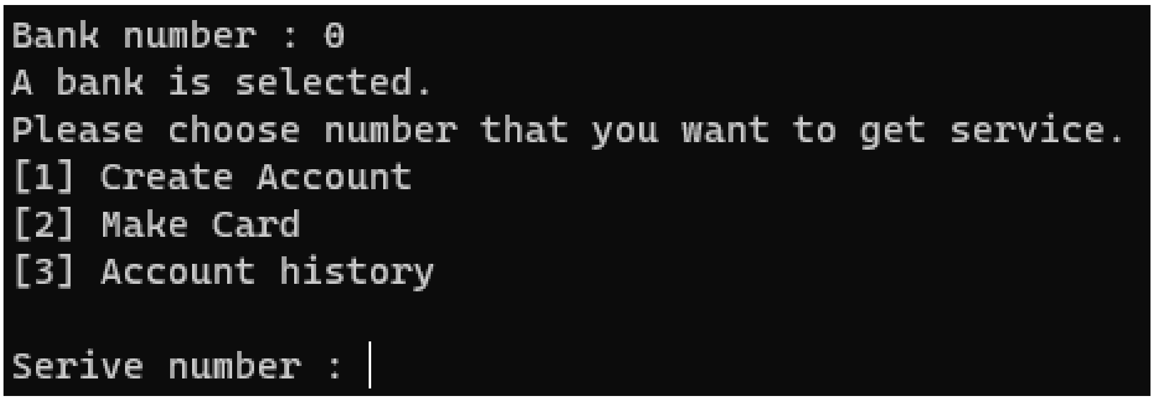
```
Please Enter the Number : 1
===================== < Bank Dup
Please write bank name : Kakao
Kakao Bank is created.
```



```
Please Enter the Number : 1
===================== < Bank Dupl
Please write bank name : kakao
kakao is already exiested!
Please write bank name :
```

은행을 생성하는 기능이다. 은행을 생성 시 먼저 생성된 은행의 이름을 기준으로 중복 여부를 파악해 같은 이름의 여러 개의 은행 생성을 방지한다. 예를 들어, Kakao를 생성 후 대소문자를 다르게 하여 kakao로 생성을 시도하면 이름이 중복인 은행이 있다고 경고를 하며 은행이 생성되지 않는다.

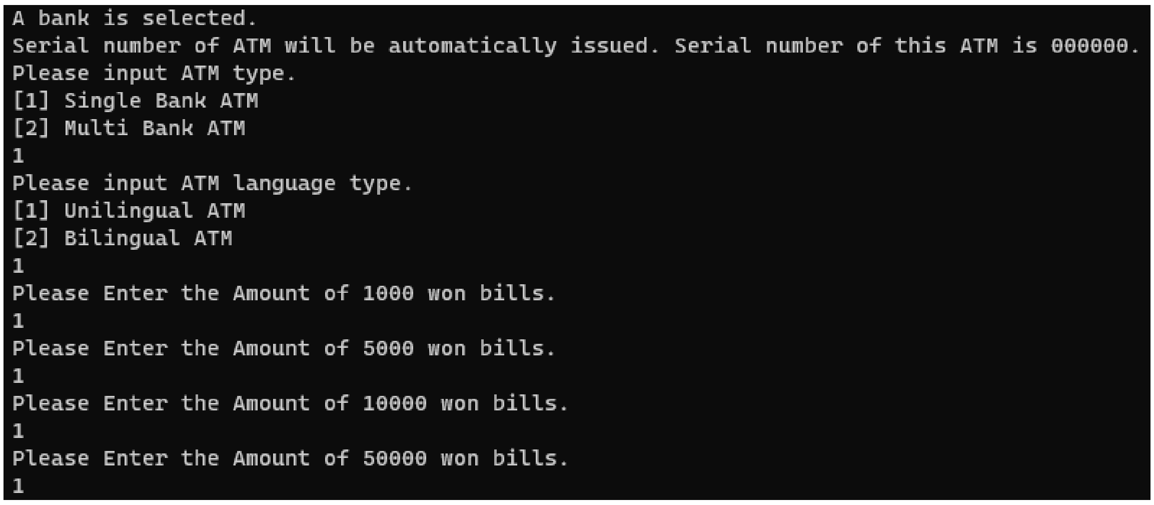
2. Banking Service



```
Bank number : 0
A bank is selected.
Please choose number that you want to get service.
[1] Create Account
[2] Make Card
[3] Account history

Serive number : 
```

원하는 대상 은행을 선택한 후 해당 은행의 계좌 생성, 카드 생성, 계좌 기록 조회 기능을 제공한다. 계좌 생성에 이름, 원하는 비밀번호, 초기 잔고를 입력해야 하고, 계좌를 생성한 직후 해당 계좌에 연결되는 카드도 만들 것 인지를 [Y/N]을 통해 고를 수 있다. 카드 생성은 계좌 번호를 입력하여 계좌를 확인한 후 비밀번호가 일치해야 성공적으로 카드가 생성된다. 계좌 기록 조회도 계좌 번호로 계좌를 확인한 후 비밀번호까지 일치하여야 해당 계좌의 기록이 출력된다.

3. Make ATM

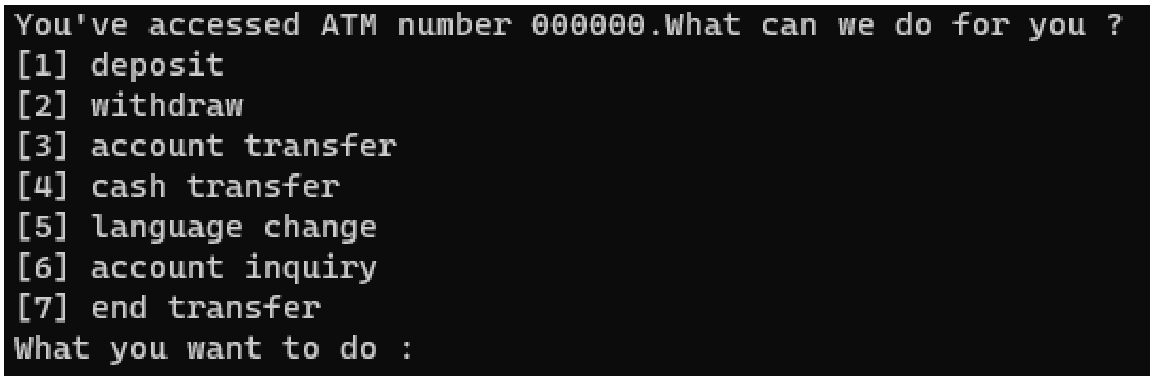


```
A bank is selected.
Serial number of ATM will be automatically issued. Serial number of this ATM is 000000.
Please input ATM type.
[1] Single Bank ATM
[2] Multi Bank ATM
1
Please input ATM language type.
[1] Unilingual ATM
[2] Bilingual ATM
1
Please Enter the Amount of 1000 won bills.
1
Please Enter the Amount of 5000 won bills.
1
Please Enter the Amount of 10000 won bills.
1
Please Enter the Amount of 50000 won bills.
1
```

주거래 은행을 선택한 후 single/multi, Uni/Bi 여부와 지폐액수별 수량을 받아서 ATM을 생성한다. 은행이 최소 1곳 이상이 존재해야 ATM을 생성할 수 있다.

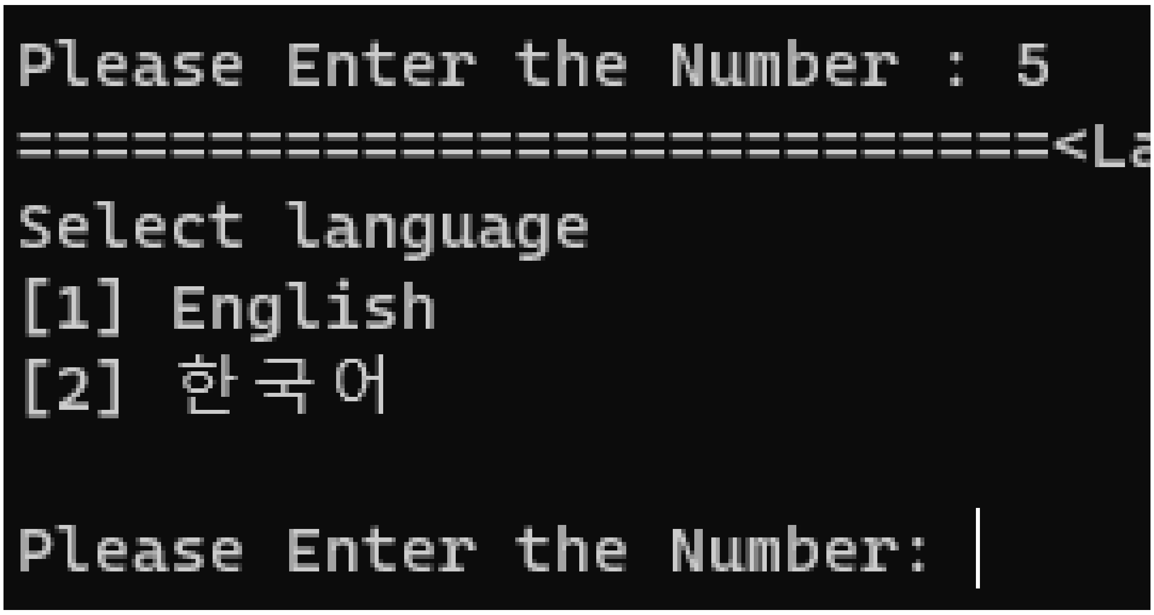
4. ATM Service

사용하고자 하는 ATM을 선택한 후 카드를 통해 계좌에 접근한다.


```
You've accessed ATM number 000000.What can we do for you ?
[1] deposit
[2] withdraw
[3] account transfer
[4] cash transfer
[5] language change
[6] account inquiry
[7] end transfer
What you want to do :
```

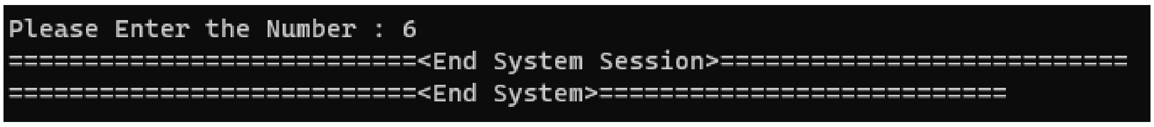
계좌의 올바른 비밀번호를 입력하면 입금, 출금, 송금 등의 다양한 기능을 사용할 수 있다.

5. Change Language Setting


```
Please Enter the Number : 5
============================<La
Select language
[1] English
[2] 한국어

Please Enter the Number:
```

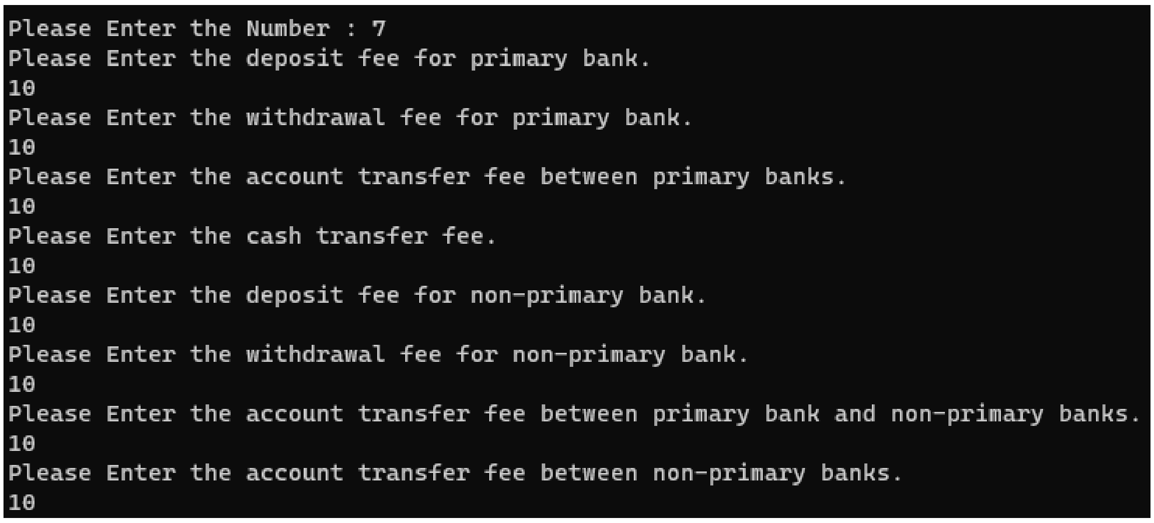
메인 세션의 언어를 한국어 또는 영어로 선택할 수 있다.

6. Shut Down the Bank System Service


```
Please Enter the Number : 6
===========================<End System Session>===========================
===========================<End System>===========================
```

메인 세션을 완전히 종료한다.

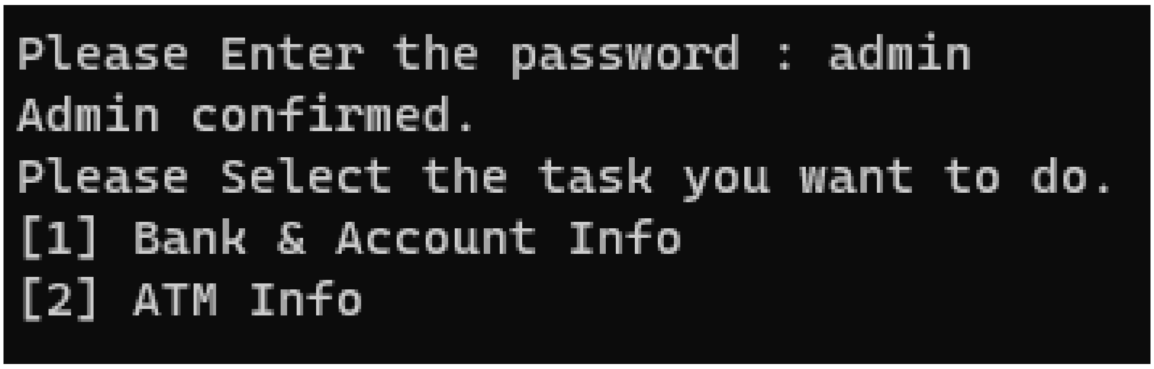
7. Fee Configuration



```
Please Enter the Number : 7
Please Enter the deposit fee for primary bank.
10
Please Enter the withdrawal fee for primary bank.
10
Please Enter the account transfer fee between primary banks.
10
Please Enter the cash transfer fee.
10
Please Enter the deposit fee for non-primary bank.
10
Please Enter the withdrawal fee for non-primary bank.
10
Please Enter the account transfer fee between primary bank and non-primary banks.
10
Please Enter the account transfer fee between non-primary banks.
10
```

다양한 조건의 수수료 금액을 지정할 수 있다.

8. Admin



```
Please Enter the password : admin
Admin confirmed.
Please Select the task you want to do.
[1] Bank & Account Info
[2] ATM Info
```

관리자 비밀번호를 통해 접근하여 모든 은행과 계좌 정보 또는 모든 ATM 정보에 접근할 수 있다.

5. Final Version of Source Code

```
// Initialization
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <exception>
#include <iomanip>
#include <typeinfo>
```

```
using namespace std;

class Bank;
class ATM;

//---------------------------------------------------------------Account Class Declarati
on
class Account {
private:
  Bank* bank;
  int account_id;
  string bank_name;
  string user_name;
  static int static_account_counter;
  static int static_card_counter;
  string account_number;
  vector<string>access_cards;
  string password;
  int avaliable_funds;
  string transaction_histories; // TransactionID, CardNumber, TransactionTypes, Amoun
t, TransactionSpecificInformation
protected:
public:
  Account(Bank* input_bank, string input_user_name, string input_password, int initial
_fund);
  ~Account();
  string getBankName();
  string getUserName();
  string getPassword();
  void setPassword(string new_password);
  string getAccountNumber();
  string makeCard();
  int checkFunds();
  void deposit(int input_money);
  void withdraw(int output_money);
  void updateHistory(string TransactionID, string CardNumber, string TransactionTypes,
string Amount, string TransactionSpecificInformation);
  void printHistory();
  vector<string> getCardNumber();
  int getAvailableFund();
  int getAccountID();
};

//------------------------------------------------------------------Bank Class Declarati
on
class Bank {
private:
  static int static_bank_counter;
  int bank_id;
  string bank_name;
  vector<Account*>accounts;
  vector<Bank*>* blist;
  vector<ATM*>* alist;
protected:
public:
  void printNow();
  void Qsearch(string* str);
  int no_error(int language_setting);
  int no_error_range(int language_setting, int min, int max);
  Bank(string name, vector<Bank*>* blist, vector<ATM*>* alist);
  ~Bank();
  int getBankId();
  string getBankName();
  void deposit2ATM(ATM* target_ATM, int numOf1000, int numOf5000, int numOf10000, int
numOf50000);
  void create_account(int language_setting);
  Account* search_account_number(int language_setting);
  Account* search_account_number_BankSearch(string input_account, int language_settin
g);
  Account* search_account_card(int language_setting);
  Account* search_account_card_BankSearch(string input_card, int language_setting);
  vector<Account*> get_account();
  void makeCard_session(int language_setting);
};

//------------------------------------------------------------------ATM Class Declarati
on
class ATM {
private:
protected:
  string serial_number;
  string admin_card;
  string language_setting = "English";
  bool lang_setting = false;
  int* cash_storage[4]; // #1000, #5000, #10000, #50000 // initial fund
  static int static_ATM_counter;
  static int static_transaction_counter;
  string transaction_histories;
  int type;
  vector<Bank*>* blist;
  vector<ATM*>* alist;
  Bank* primary_bank;
  int language_available;
public:
  void Qsearch(string* str);
  // primary bank name / serial numbeer / type : single or multi bank / language : un
i, bi / initial fund
  ATM(Bank* input_primary_bank, string input_serial_number, int input_type, int input_
lanuage_available, int* initial_fund[], vector<Bank*>* blist, vector<ATM*>* alist);
  ~ATM();
  void session(vector<Bank*> bank_list);
  vector<string> transaction(Account* a, vector<Bank*> bank_list, string CardNumber, i
nt* with_counter);
  void languageChange();
  virtual int deposit(Account* a) = 0;
  virtual int withdraw(Account* a) = 0;
  virtual int account_transfer(Account* a, Account* b) = 0;
  virtual int cash_transfer(Account* b) = 0;
  void see_transaction_history();
  bool user_authorization(Account* a);
  void add_cash(int cash1000, int cash5000, int cash10000, int cash50000);
  string getSerial();
  Bank* getPrimary();
  void make_history(vector<string> rec);
  void display_transaction(vector<string> rec);
  void display_transaction_short(vector<string> rec);
  virtual Account* card2account(string card, vector<Bank*> bank_list) = 0;
  virtual Account* num2account(string num, vector<Bank*> bank_list) = 0;
  virtual string getType() = 0;
  string getLangType();
  int get1000();
  int get5000();
  int get10000();
  int get50000();
  int no_error(bool language_setting);
  int no_error_range(bool language_setting, int min, int max);
  void printNow();
};

//-----------------------------------------------------Single Bank ATM Class Declarati
on
class Single : public ATM {
private:
  //Fee my_fee;
  int* fee_list[4];
protected:
public:
  Single();
  Single(Bank* input_primary_bank, string input_serial_number, int input_lanuage_avail
able, int* initial_fund[], int* fees[4], vector<Bank*>* blist, vector<ATM*>* alist);
  ~Single();
  int deposit(Account* a);
  int withdraw(Account* a);
  int account_transfer(Account* a, Account* b);
  int cash_transfer(Account* b);
  Account* card2account(string card, vector<Bank*> bank_list);
  Account* num2account(string num, vector<Bank*> bank_list);
  string getType();
};

//------------------------------------------------------Multi Bank ATM Class Declarati
on
class Multi : public ATM {
private:
  //Fee my_fee;
  int* fee_list[4];
  int* multi_fee_list[4];
protected:
public:
  Multi();
  Multi(Bank* input_primary_bank, string input_serial_number, int input_lanuage_availa
ble, int* initial_fund[], int* fees[4], int* mfees[4], vector<Bank*>* blist, vector<AT
M*>* alist);
  ~Multi();
  int deposit(Account* a);
  int withdraw(Account* a);
  int account_transfer(Account* a, Account* b);
  int cash_transfer(Account* b);
  Account* card2account(string card, vector<Bank*> bank_list);
  Account* num2account(string num, vector<Bank*> bank_list);
  string getType();
};

//---------------------------------------------------------Account Class Member Definiti
on
int Account::static_card_counter = 0;
int Account::static_account_counter = 0;

Account::Account(Bank* input_bank, string input_user_name, string input_password, int
initial_fund) {
  this->bank = input_bank;
  this->bank_name = input_bank->getBankName();
  this->user_name = input_user_name;
  this->password = input_password;
  this->avaliable_funds = initial_fund;
  string temp_bank_code;
  for (int i = 0; i < 3 - to_string(input_bank->getBankId()).size(); i++) {
    temp_bank_code += "0";
  }
  temp_bank_code += to_string(input_bank->getBankId());
  this->static_account_counter += 1;
  this->account_id = this->static_account_counter;
  string temp_account_code;
  for (int i = 0; i < 6 - to_string(this->account_id).size(); i++) {
    temp_account_code += "0";
  }
  temp_account_code += to_string(this->account_id);
  this->account_number = "000-" + temp_bank_code + "-" + temp_account_code;
  this->transaction_histories = input_user_name + ".txt";
  this->updateHistory("0", "None", "0", to_string(avaliable_funds), "Generate New Acco
unt"); // TransactionID, CardNumber, TransactionTypes, Amount, TransactionSpecificInfo
rmation
}

Account::~Account() {
  this->bank_name = "";
  this->user_name = "";
  this->password = "";
  this->avaliable_funds = 0;
  this->updateHistory("0", "None", "0", to_string(avaliable_funds), "Delete Account");
// TransactionID, CardNumber, TransactionTypes, Amount, TransactionSpecificInformation
}

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

string Account::getUserName() {
  return this->user_name;
}

string Account::getPassword() {
  return this->password;
```

```
}

void Account::setPassword(string new_password) {
  this->password = new_password;
}

string Account::getAccountNumber() {
  return this->account_number;
}

string Account::makeCard() {
  string temp_bank_code;
  for (int i = 0; i < 4 - to_string(this->bank->getBankId()).size(); i++) {
    temp_bank_code += "0";
  }
  temp_bank_code += to_string(this->bank->getBankId());
  // this->static_account_counter += 1;
  string temp_account_code;
  for (int i = 0; i < 4 - to_string(this->getAccountID()).size(); i++) {
    temp_account_code += "0";
  }
  temp_account_code += to_string(this->getAccountID());
  this->static_card_counter += 1;
  string temp_card_code;
  for (int i = 0; i < 4 - to_string(static_card_counter).size(); i++) {
    temp_card_code += "0";
  }
  temp_card_code += to_string(static_card_counter);
  string card_number = "0000-" + temp_bank_code + "-" + temp_account_code + "-" + temp
_card_code;
  this->access_cards.push_back(card_number);
  return card_number;
}

int Account::checkFunds() {
  return this->avaliable_funds;
}

void Account::deposit(int input_money) {
  this->avaliable_funds = this->avaliable_funds + input_money;
}

void Account::withdraw(int output_money) {
  this->avaliable_funds = this->avaliable_funds - output_money;
}

void Account::updateHistory(string TransactionID, string CardNumber, string Transactio
nTypes, string Amount, string TransactionSpecificInformation) {
  vector<string>new_history = { TransactionID, CardNumber, TransactionTypes, Amount, T
ransactionSpecificInformation }; // TransactionID, CardNumber, TransactionTypes, Amoun
t, TransactionSpecificInformation
  int len = static_cast<int>(new_history.size());
  ofstream writeFromFile(this->transaction_histories, ios::app);
  for (int i = 0; i < len; ++i) {
    string tmp = new_history[i];
    if (i != len - 1) {
      tmp += ", ";
    }
    writeFromFile << tmp;
  }
  writeFromFile << "\n";
}

void Account::printHistory() {
  ifstream readFile(this->transaction_histories);
  if (readFile.is_open()) {
    string str;
    while (readFile) {
      getline(readFile, str);
      cout << str << endl;
    }
  }
  else {
    cout << "해당하는 파일이 없습니다." << endl;
  }
}

vector<string> Account::getCardNumber() {
  return access_cards;
}

int Account::getAvailableFund() {
  return avaliable_funds;
}

int Account::getAccountID() {
  return account_id;
}

//----------------------------------------------------------Bank Class Member Definiti
on
int Bank::static_bank_counter = 0;

void Bank::printNow() {
  for (int i = 0; i < (*blist).size(); i++) {
    for (int j = 0; j < (*blist)[i]->get_account().size(); j++) {
      cout << "Account[" << (*blist)[i]->get_account()[j]->getUserName() << "] : " <<
(*blist)[i]->get_account()[j]->checkFunds() << "/" << (*blist)[i]->get_account()[j]->g
etAccountNumber() << "/" << (*blist)[i]->get_account()[j]->getBankName() << endl;
    }
  }
  for (int i = 0; i < (*alist).size(); i++) {
    cout << "ATM " << "[" << i << "] " << "Primary Bank : " << (*alist)[i]->getPrimary
()->getBankName() << " " << (*alist)[i]->getType() << " " << (*alist)[i]->getLangType
() << endl;
    cout << "Remaing cash : " << 1000 * (*alist)[i]->get1000() + 5000 * (*alist)[i]->g
et5000() + 10000 * (*alist)[i]->get10000() + 50000 * (*alist)[i]->get50000() << " (100
0 : " << (*alist)[i]->get1000() << ", 5000 : " << (*alist)[i]->get5000() << ", 10000 :
" << (*alist)[i]->get10000() << ", 50000 : " << (*alist)[i]->get50000() << ")" << end
l;
    // cout << "ATM [" << ATM_list[i]->getSerial() << "] Remaing cash : " << 1000 * AT
M_list[i]->get1000() + 5000 * ATM_list[i]->get5000() + 10000 * ATM_list[i]->get10000()
```

```
+ 50000 * ATM_list[i]->get50000() << " (1000 : " << ATM_list[i]->get1000() << ", 5000
: " << ATM_list[i]->get5000() << ", 10000 : " << ATM_list[i]->get10000() << ", 50000 :
" << ATM_list[i]->get50000() << ")" << endl;
  }
  cout << endl;
  cout << "Please re-enter here : ";
  return;
}

void Bank::Qsearch(string* str) {
  while (true) {
    cin >> *str;
    if (*str == "Q" || *str == "q") {
      printNow();
    }
    else {
      return;
    }
  }
}

int Bank::no_error(int language_setting) {
  int temp = 0;
  while (true) {
    string abc = "";
    //cin >> abc;
    while (true) {
      cin >> abc;
      if (abc == "Q" || abc == "q") {
        printNow();
      }
      else {
        break;
      }
    }
    if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
("\n") != string::npos) {
      if (language_setting == 1) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
    if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
      temp = stoi(abc);
      if (temp >= 0) { return temp; }
      else {
        if (language_setting == 1) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 2) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
    if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
      if (language_setting == 1) {
        cout << "[Error] Input out of range. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
  }
}

int Bank::no_error_range(int language_setting, int min, int max) {
  int temp = 0;
  while (true) {
    string abc = "";
    //cin >> abc;
    while (true) {
      cin >> abc;
      if (abc == "Q" || abc == "q") {
        printNow();
      }
      else {
        break;
      }
    }
    if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
("\n") != string::npos) {
      if (language_setting == 1) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
    if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
      temp = stoi(abc);
      if (temp > min - 1 && temp < max + 1) { return temp; }
      else {
        if (language_setting == 1) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 2) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
    if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
      if (language_setting == 1) {
        cout << "[Error] Input out of range. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
  }
}

Bank::Bank(string name, vector<Bank*>* blist, vector<ATM*>* alist) {
  this->bank_name = name;
  static_bank_counter += 1;
  this->bank_id = static_bank_counter;
  this->blist = blist;
  this->alist = alist;
  cout << this->getBankName() << " Bank is created." << endl;
}

Bank::~Bank() {
  cout << this->bank_name << " Bank is eliminated." << endl;
}

int Bank::getBankId() {
  return this->bank_id;
}

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

void Bank::deposit2ATM(ATM* target_ATM, int numOf1000, int numOf5000, int numOf10000,
int numOf50000) {
  target_ATM->add_cash(numOf1000, numOf5000, numOf10000, numOf50000);
}

void Bank::makeCard_session(int language_setting) {
  cout << "=================== < Card Create Session > ===================" << endl;
  if (accounts.size() == 0) {
    if (language_setting == 1) { cout << "There is no account. So making card is not a
vailable." << endl; }
    if (language_setting == 2) { cout << "은행에 존재하는 계좌가 없어 카드 생성을 수행할 수 없습니
다." << endl; }
    return;
  }
  int c = 0;
  Account* account = search_account_number(language_setting);
  if (account == NULL) { c = 1; }
  if (c == 0) {
    string input_password;
    if (language_setting == 1) { cout << "To make card, please write password." << end
l; }
    if (language_setting == 2) { cout << "카드를 만들기 위해 비밀번호를 입력해주세요." << endl;
}
    int a = 3;
    while (true) {
      if (a == 0) {
        if (language_setting == 1) { cout << "You write wrong password 3 times. " << e
ndl; }
        if (language_setting == 2) { cout << "비밀번호 3회 틀렸습니다." << endl; }
        cout << "==================== < Card Create Session End! > ===================
=" << endl;
        break;
      }
      if (language_setting == 1) {
        cout << a << " attempts left." << endl;
        cout << "Password : ";
      }
      if (language_setting == 2) {
        cout << a << " 회 남음." << endl;
        cout << "비밀번호 : ";
      }
      string* pinput_password = &input_password;
      Qsearch(pinput_password);
      a--;
      if (account->getPassword() == input_password) {
        if (language_setting == 1) { cout << "Correct password." << endl; }
        if (language_setting == 2) { cout << "옳은 비밀번호." << endl; }
        string now_created_card_number;
        now_created_card_number = account->makeCard();
        if (language_setting == 1) {
          cout << "Card is created." << endl;
          cout << "Card Number : " << now_created_card_number << endl;
          cout << "This is card number list that connected to your account." << endl;
        }
        if (language_setting == 2) {
          cout << "카드가 생성되었습니다." << endl;
          cout << "카드 번호 : " << now_created_card_number << endl;
          cout << "계좌에 연결된 카드 번호 목록입니다." << endl;
        }
        vector <string > card_list = account->getCardNumber();
        for (int i = 0; i < card_list.size(); i++) {
          cout << card_list[i] << endl;
        }
        break;
      }
    }
  }
  cout << "==================== < Card Create Session End! > ====================" <<
endl;
}

void Bank::create_account(int language_setting) {
  cout << "==================== < Account Create Session > ====================" << en
dl;
  string input_user_name;
  string* pinput_user_name = &input_user_name;
  string account_number;
  string input_password;
  string* pinput_password = &input_password;
  if (language_setting == 1) {
    cout << this->getBankName() << "Bank. To create account. please write name, passwo
rd and initial fund." << endl;
    cout << "Name : ";
    // cin >> input_user_name;
    Qsearch(pinput_user_name);
    cout << "Password : ";
    // cin >> input_password;
    Qsearch(pinput_password);
    cout << "Initial fund : ";
  }
  if (language_setting == 2) {
    cout << this->getBankName() << "은행. 계좌를 생성하기 위해 이름, 계좌, 초기 자금을 입력해주세
요." << endl;
    cout << "예금주 : ";
    // cin >> input_user_name;
    Qsearch(pinput_user_name);
    cout << "비밀번호 : ";
    // cin >> input_password;
    Qsearch(pinput_password);
    cout << "초기자금 : ";
  }
  int initial_fund;
  // cin >> initial_fund;
  initial_fund = no_error(language_setting);
  Account* new_account = new Account(this, input_user_name, input_password, initial_fu
nd);
  accounts.push_back(new_account);
  if (language_setting == 1) {
    cout << "Account is created." << endl;
    cout << "Bank : " << new_account->getBankName() << endl;
    cout << "Owner : " << new_account->getUserName() << endl;
    cout << "Account number : " << new_account->getAccountNumber() << endl;
    cout << "Password : " << new_account->getPassword() << endl;
    cout << "Available fund : " << new_account->getAvailableFund() << endl;
    cout << "Do you want to make card? [Agree Y / Disagree N] : ";
  }
  if (language_setting == 2) {
    cout << "계좌가 개설되었습니다." << endl;
    cout << "은행 : " << new_account->getBankName() << endl;
    cout << "예금주 : " << new_account->getUserName() << endl;
    cout << "계좌번호 : " << new_account->getAccountNumber() << endl;
    cout << "비밀번호 : " << new_account->getPassword() << endl;
    cout << "계좌잔고 : " << new_account->getAvailableFund() << endl;
    cout << "카드를 생성하시겠습니까? [동의 Y / 비동의 N] : ";
  }
  string agreement;
  string* pagreement = &agreement;
  // cin >> agreement;
  while (true) {
    // cin >> agreement;
    Qsearch(pagreement);
    if (agreement != "n" && agreement != "N" && agreement != "y" && agreement != "Y")
{
      if (language_setting == 1) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "[Agree Y / Disagree N] : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "[동의 Y / 비동의 N] : ";
      }
    }
    else { break; }
  }
  if (agreement == "n" || agreement == "N") { cout << "==================== < Card Cre
ate Session End! > ====================" << endl; }
  if (agreement == "y" || agreement == "Y") {
    cout << "==================== < Card Create Session > ====================" << end
l;
    string now_created_card_number;
    now_created_card_number = new_account->makeCard();

    if (language_setting == 1) {
      cout << "Card is created." << endl;
      cout << "Card number : " << now_created_card_number << endl;
      cout << "This is card number list that connected to your account." << endl;
    }
    if (language_setting == 2) {
      cout << "카드가 생성되었습니다." << endl;
      cout << "카드 번호 : " << now_created_card_number << endl;
      cout << "계좌에 연결된 카드 번호 목록입니다." << endl;
    }
    vector <string> card_list = new_account->getCardNumber();
    for (int i = 0; i < card_list.size(); i++) {
      cout << card_list[i] << endl;
    }
    cout << "==================== < Card Create Session End! > ====================" <
< endl;
  }
  cout << "==================== < Account Create Session End! > ===================="
<< endl;
  //return new_account;
}

Account* Bank::search_account_number(int language_setting) {
  cout << "==================== < Account Number Search Session > ===================
=" << endl;
  vector<Account*> accounts_list = get_account();
  int a = 3;
  while (true) {
    if (a == 0) {
      if (language_setting == 1) { cout << this->getBankName() << " Bank cannot find y
our account." << endl; }
      if (language_setting == 2) { cout << this->getBankName() << " 은행이 계좌를 찾지 못했
습니다." << endl; }
      break;
    }
    if (language_setting == 1) {
      cout << a << " attempts left." << endl;
      cout << "Account number : ";
    }
    if (language_setting == 2) {
      cout << a << " 회 남음." << endl;
      cout << "계좌번호 : ";
    }
    a--;
    string input_account_number;
    string* pinput_account_number = &input_account_number;
    // cin >> input_account_number;
    Qsearch(pinput_account_number);
    for (int i = 0; i < accounts_list.size(); i++) {
      if (accounts_list[i]->getAccountNumber() == input_account_number) {
        if (language_setting == 1) {
          cout << this->getBankName() << " Bank find your account." << endl;
          cout << "Bank : " << accounts_list[i]->getBankName() << endl;
          cout << "Owner : " << accounts_list[i]->getUserName() << endl;
          cout << "Account : " << accounts_list[i]->getAccountNumber() << endl;
        }
        if (language_setting == 2) {
          cout << this->getBankName() << " 은행이 계좌를 찾았습니다." << endl;
          cout << "은행 : " << accounts_list[i]->getBankName() << endl;
          cout << "예금주 : " << accounts_list[i]->getUserName() << endl;
          cout << "계좌번호 : " << accounts_list[i]->getAccountNumber() << endl;
        }
        cout << "==================== < Account Number Search Session End! > =========
===========" << endl;
        return accounts_list[i];
      }
    }
  }
  if (language_setting == 1) { cout << "You write wrong account number 3 times." << en
dl; }
  if (language_setting == 2) { cout << "잘못된 계좌 번호를 3회 입력했습니다." << endl; }
  cout << "==================== < Account Number Search Session End! > ===============
=====" << endl;
  return NULL;
}

Account* Bank::search_account_number_BankSearch(string input_account, int language_set
ting) {
  vector<Account*> accounts_list = get_account();
  string input_account_number = input_account;
  for (int i = 0; i < accounts_list.size(); i++) {
    if (accounts_list[i]->getAccountNumber() == input_account_number) {
      if (language_setting == 1) {
        cout << this->getBankName() << " Bank find your account." << endl;
        cout << "Bank : " << accounts_list[i]->getBankName() << endl;
        cout << "Owner : " << accounts_list[i]->getUserName() << endl;
        cout << "Account : " << accounts_list[i]->getAccountNumber() << endl;
      }
      if (language_setting == 2) {
        cout << this->getBankName() << " 은행이 계좌를 찾았습니다." << endl;
        cout << "은행 : " << accounts_list[i]->getBankName() << endl;
        cout << "예금주 : " << accounts_list[i]->getUserName() << endl;
        cout << "계좌번호 : " << accounts_list[i]->getAccountNumber() << endl;
      }
      return accounts_list[i];
    }
  }
  return NULL;
}

Account* Bank::search_account_card(int language_setting) {
  cout << "=================== < Account Card Search Session > ==================="
<< endl;
  vector<Account*> accounts_list = get_account();
  if (language_setting == 1) { cout << this->getBankName() << " Bank. Please write car
d number." << endl; }
  if (language_setting == 2) { cout << this->getBankName() << " 은행. 카드 번호를 입력해주세
요." << endl; }
  string input_card_number;
  string* pinput_card_number = &input_card_number;
  if (language_setting == 1) {
    cout << "Card number : ";
    Qsearch(pinput_card_number);
  }
  if (language_setting == 2) {
    cout << "카드 번호 : ";
    Qsearch(pinput_card_number);
  }
  for (int i = 0; i < accounts_list.size(); i++) {
    vector<string> card_list = accounts_list[i]->getCardNumber();
    for (int j = 0; j < card_list.size(); j++) {
      if (card_list[j] == input_card_number) {
        if (language_setting == 1) {
          cout << this->getBankName() << " Bank find your account." << endl;
          cout << "Bank : " << accounts_list[i]->getBankName() << endl;
          cout << "Owner : " << accounts_list[i]->getUserName() << endl;
          cout << "Account : " << accounts_list[i]->getAccountNumber() << endl;
        }
        if (language_setting == 2) {
          cout << this->getBankName() << " 은행이 계좌를 찾았습니다." << endl;
          cout << "은행 : " << accounts_list[i]->getBankName() << endl;
          cout << "예금주 : " << accounts_list[i]->getUserName() << endl;
          cout << "계좌번호 : " << accounts_list[i]->getAccountNumber() << endl;
        }
        cout << "=================== < Account Card Search Session End! > ===========
=========" << endl;
        return accounts_list[i];
      }
    }
  }
  if (language_setting == 1) { cout << this->getBankName() << " Bank cannot find your
account." << endl; }
  if (language_setting == 2) { cout << this->getBankName() << " 은행이 계좌를 찾지 못했습니
다." << endl; }
```

```
  cout << "=================== < Account Card Search Session End! > =================
===" << endl;
  return NULL;
}

Account* Bank::search_account_card_BankSearch(string input_card, int language_setting)
{
  vector<Account*> accounts_list = get_account();
  string input_card_number = input_card;
  for (int i = 0; i < accounts_list.size(); i++) {
    vector<string> card_list = accounts_list[i]->getCardNumber();
    for (int j = 0; j < card_list.size(); j++) {
      if (card_list[j] == input_card_number) {
        if (language_setting == 1) {
          cout << this->getBankName() << " Bank find your account." << endl;
          cout << "Bank : " << accounts_list[i]->getBankName() << endl;
          cout << "Owner : " << accounts_list[i]->getUserName() << endl;
          cout << "Account : " << accounts_list[i]->getAccountNumber() << endl;
        }
        if (language_setting == 2) {
          cout << this->getBankName() << " 은행이 계좌를 찾았습니다." << endl;
          cout << "은행 : " << accounts_list[i]->getBankName() << endl;
          cout << "예금주 : " << accounts_list[i]->getUserName() << endl;
          cout << "계좌번호 : " << accounts_list[i]->getAccountNumber() << endl;
        }
        return accounts_list[i];
      }
    }
  }
  return NULL;
}

vector<Account*> Bank::get_account() {
  return accounts;
}

//----------------------------------------------------------ATM Class Member Definiti
on
int ATM::static_ATM_counter = 0;
int ATM::static_transaction_counter = 0;

void ATM::Qsearch(string* str) {

  while (true) {
    cin >> *str;
    if (*str == "Q" || *str == "q") {
      printNow();
    }
    else {
      return;
    }
  }
}

int ATM::no_error(bool language_setting) {
  int temp = 0;
```

```
    while (true) {

      string abc = "";
      // cin >> abc;
      while (true) {
        cin >> abc;
        if (abc == "Q" || abc == "q") {
          printNow();
        }
        else {
          break;
        }
      }

      if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
  ("\n") != string::npos) {
        if (language_setting == 0) {
          cout << "[Error] An input error has occurred. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 1) {
          cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }

      if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
        temp = stoi(abc);
        if (temp >= 0) { return temp; }
        else {
          if (language_setting == 0) {
            cout << "[Error] Input out of range. Please write again." << endl;
            cout << "Please Enter the Number : ";
          }
          if (language_setting == 1) {
            cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
            cout << "숫자를 입력해주세요 : ";
          }
          continue;
        }
      }
      if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
        if (language_setting == 0) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 1) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
  }
```

```
int ATM::no_error_range(bool language_setting, int min, int max) {
  int temp = 0;
  while (true) {

    string abc = "";
    // cin >> abc;
    while (true) {
      cin >> abc;
      if (abc == "Q" || abc == "q") {
        printNow();
      }
      else {
        break;
      }
    }

    if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
("\n") != string::npos) {
      if (language_setting == 0) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 1) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }

    if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
      temp = stoi(abc);
      if (temp > min - 1 && temp < max + 1) { return temp; }
      else {
        if (language_setting == 0) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 1) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
    if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
      if (language_setting == 0) {
        cout << "[Error] Input out of range. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 1) {
        cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
  }
}

void ATM::languageChange() {
  if (language_available == 1) {
    cout << "unilingual ATM cannot change language!" << endl;
    return;
  }
  if (lang_setting == false) { cout << "Please choose the language setting." << endl;
}
  else { cout << "언어를 선택해 주세요." << endl; }
  cout << "[1] English" << endl;
  cout << "[2] 한국어" << endl;
  if (lang_setting == false) { cout << "Input : " << endl; }
  else { cout << "입력 : " << endl; }

  int input_language_setting = no_error_range(lang_setting, 1, 2);

  if (input_language_setting == 2) {
    //this->language_setting = "Korean";
    this->lang_setting = true;
  }
  if (input_language_setting == 1) {
    //this->language_setting = "English";
    this->lang_setting = false;
  }
  return;
}

ATM::ATM(Bank* input_primary_bank, string input_serial_number, int input_type, int inp
ut_lanuage_available, int* initial_fund[], vector<Bank*>* blist, vector<ATM*>* alist)
{
  this->primary_bank = input_primary_bank;
  this->serial_number = input_serial_number;
  this->type = input_type;
  this->language_available = input_lanuage_available;
  this->transaction_histories = input_serial_number + ".txt";
  this->admin_card = input_primary_bank->getBankName() + "admin";
  this->blist = blist;
  this->alist = alist;

  for (int i = 0; i < 4; ++i) {
    this->cash_storage[i] = initial_fund[i];
  }
}

ATM::~ATM() {
  this->primary_bank = nullptr;
  this->serial_number = "";
  this->type = 0;
  this->language_available = 0;

  for (int i = 0; i < 4; ++i) {
    this->cash_storage[i] = 0;
  }
}
```

```
void ATM::session(vector<Bank*> bank_list) {
  if (this->lang_setting == true) { cout << "카드를 삽입해 주세요." << endl << "카드 번호 :
"; }
  if (this->lang_setting == false) { cout << "Please insert your card." << endl << "Ca
rd number : "; }
  string cardinsert = "";
  string* pcardinsert = &cardinsert;
  //cin >> cardinsert;

  Qsearch(pcardinsert);

  if (cardinsert == this->admin_card) {
    see_transaction_history();
    return;
  }
  Account* acc = card2account(cardinsert, bank_list);
  if (acc == nullptr) {
    if (this->lang_setting == 1) { cout << "잘못된 카드가 입력되었습니다. 카드를 반환합니다." <<
endl; }
    if (this->lang_setting == 0) { cout << "Wrong card has inserted. Card will be retu
rn." << endl; }
    return;
  }
  if (this->user_authorization(acc) == false) {
    if (this->lang_setting == 1) { cout << "비밀번호 입력에 3회 실패하셨습니다. 거래를 종료합니
다." << endl; }
    if (this->lang_setting == 0) { cout << "Failed to enter password 3 times. Transact
ion will be canceled." << endl; }
    return;
  }
  int* with_counter = new int(0);
  vector<vector<string>> record;
  for (int i = 0;; i++) {
    record.push_back(transaction(acc, bank_list, cardinsert, with_counter));
    if (record.at(i).at(2) == "Termination") {
      break;
    }
  }
  delete with_counter;
  for (int i = 0; i < record.size(); i++) {
    display_transaction_short(record.at(i));
  }
  return;
}

vector<string> ATM::transaction(Account* a, vector<Bank*> bank_list, string CardNumbe
r, int* with_counter) {
  vector<string> rec; //transactionID, card number, transaction type, success or failu
re, amount, note
  rec.push_back(to_string(++static_transaction_counter));
  rec.push_back(CardNumber);
  int amount = 0;
  string note = "";

  if (this->lang_setting == 1) { cout << this->getSerial() << "번 ATM에 접속하셨습니다. 무슨
작업을 도와드릴까요?" << endl; }
```

```
  else if (this->lang_setting == 0) { cout << "You've accessed ATM number " << this->g
etSerial() << ".What can we do for you ? " << endl; }
  if (this->lang_setting == 1) { cout << "[1] 입금" << endl << "[2] 출금" << endl << "
[3] 계좌 송금" << endl << "[4] 현금 송금" << endl << "[5] 언어 변경" << endl << "[6] 계좌 조
회" << endl << "[7] 거래 종료" << endl << "원하는 작업 : "; }
  else if (this->lang_setting == 0) { cout << "[1] deposit" << endl << "[2] withdraw"
<< endl << "[3] account transfer" << endl << "[4] cash transfer" << endl << "[5] langu
age change" << endl << "[6] account inquiry" << endl << "[7] end transfer" << endl <<
"What you want to do : "; }

  int selection = no_error_range(lang_setting, 1, 7);
  if (selection == 1) {
    rec.push_back("Deposit");
    amount = deposit(a);
  }
  else if (selection == 2 && *with_counter < 3) {
    rec.push_back("Withdraw");
    amount = withdraw(a);
    *with_counter += 1;
  }
  else if (selection == 2 && *with_counter >= 3) {
    rec.push_back("Withdraw");
    if (this->lang_setting == 1) { cout << "출금은 한 세션에 3회까지만 가능합니다!!" << endl;
}
    else { cout << "The maximum number of withdrawals per each session is 3!!" << end
l; }
    amount = -1;
  }
  else if (selection == 3) {
    rec.push_back("account transfer");
    if (this->lang_setting == 1) { cout << "송금할 계좌의 번호를 입력해 주세요 : "; }
    if (this->lang_setting == 0) { cout << "Please enter the number of the account you
want to transfer : "; }
    string numinsert1 = "";
    string* pnuminsert1 = &numinsert1;
    Qsearch(pnuminsert1);
    Account* b = num2account(numinsert1, bank_list);
    if (b == nullptr) {
      if (this->lang_setting == 1) { cout << "잘못된 계좌 번호를 입력하셨습니다. 거래를 종료합니
다." << endl; }
      if (this->lang_setting == 0) { cout << "Invalid account number. Transaction will
be canceled." << endl; }
      amount = -1;
    }
    else {
      amount = account_transfer(a, b);
      note = b->getAccountNumber();
    }
  }
  else if (selection == 4) {
    rec.push_back("cash transfer");
    if (this->lang_setting == 1) { cout << "송금할 계좌의 번호를 입력해 주세요 : "; }
    if (this->lang_setting == 0) { cout << "Please enter the number of the account you
want to transfer : "; }
    string numinsert2 = "";
    string* pnuminsert2 = &numinsert2;
    Qsearch(pnuminsert2);
    Account* b = num2account(numinsert2, bank_list);
    if (b == nullptr) {
      if (this->lang_setting == 1) { cout << "잘못된 계좌 번호를 입력하셨습니다. 거래를 종료합니
다." << endl; }
      if (this->lang_setting == 0) { cout << "Invalid account number. Transaction will
be canceled." << endl; }
      amount = -1;
    }
    else {
      amount = cash_transfer(b);
      note = b->getAccountNumber();
    }
  }
  else if (selection == 5) {
    rec.push_back("language change");
    languageChange();
    amount = 0;
  }
  else if (selection == 6) {
    rec.push_back("Account inquiry");
    a->printHistory();
    amount = 0;
  }
  else if (selection == 7) {
    rec.push_back("Termination");
    amount = 0;
  }
  else {
    if (lang_setting == true) { cout << "입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." <<
endl; }
    else { cout << "An input error has occurred. Please type the input again." << end
l; }
    rec.push_back("invalid transaction");
    rec.push_back("Failure");
    rec.push_back("-1");
  }
  if (amount == -1) { rec.push_back("Failure"); }
  else { rec.push_back("Success"); }
  rec.push_back(to_string(amount));
  rec.push_back(note);
  make_history(rec);
  a->updateHistory(rec[0], rec[1], rec[2], rec[4], rec[5]);
  return rec;
}

void ATM::add_cash(int cash1000, int cash5000, int cash10000, int cash50000) {
  *(this->cash_storage[0]) += cash1000;
  *(this->cash_storage[1]) += cash5000;
  *(this->cash_storage[2]) += cash10000;
  *(this->cash_storage[3]) += cash50000;
}

bool ATM::user_authorization(Account* a) {
  //카드 입력
  //카드 유효성 검사

  //비밀번호 입력

  //비밀번호 검사

  //3번 연속 틀리면 세션 중단되게
  if (this->lang_setting == 1) { cout << "안녕하세요, " << a->getUserName() << "." << end
l; }
  if (this->lang_setting == 0) { cout << "Hello, " << a->getUserName() << "." << endl;
}
  if (this->lang_setting == 1) { cout << "비밀번호를 3회 틀리면, 거래<가 종료됩니다." << endl;
}
  if (this->lang_setting == 0) { cout << "If you get the password wrong 3 times, the t
ransaction ends." << endl; }
  string pass = "";
  string* ppass = &pass;
  for (int i = 0; i < 3; i++) {
    if (this->lang_setting == 1) { cout << "비밀번호를 입력해 주십시오." << endl; }
    else if (this->lang_setting == 0) { cout << "Please enter your password." << endl;
}
    if (this->lang_setting == 1) { cout << "남은 입력 횟수 : " << 3 - i << "회" << endl <<
"비밀번호 : "; }
    else if (this->lang_setting == 0) { cout << "remaining : " << 3 - i << " times" <<
endl << "password : "; }
    Qsearch(ppass);
    if (a->getPassword() == pass) {
      return true;
    }
    if (this->lang_setting == 1) { cout << "비밀번호가 틀렸습니다." << endl; }
    if (this->lang_setting == 0) { cout << "Wrong password." << endl; }
  }
  return false;
}

void ATM::see_transaction_history() {
  if (this->lang_setting == 1) { cout << "관리자 카드가 입력되었습니다." << endl << "관리자 권
한으로 어떤 작업을 하시겠습니까?" << endl << "[0] 거래 내역 확인 " << endl; }
  if (this->lang_setting == 0) { cout << "Admin card has inserted." << endl << "What d
o you want to do with administrator privileges?" << endl << "[0] Transaction history "
<< endl; }
  int no = no_error_range(lang_setting, 0, 2);
  if (no == 2) {
    if (lang_setting == true) { cout << "초기 화면으로 돌아갑니다." << endl; }
    else { cout << "Returning to the initial screen." << endl; }
    return;
  }
  if (no == 1) {
    int replenish[4] = { 0,0,0,0 };
    if (this->lang_setting == 1) { cout << "보충할 1000원권의 양을 입력해 주세요." << endl; }
    if (this->lang_setting == 0) { cout << "Please enter the amount of 1,000 won bills
to be replenished." << endl; }
    replenish[0] = no_error(lang_setting);
    if (this->lang_setting == 1) { cout << "보충할 5000원권의 양을 입력해 주세요." << endl; }
    if (this->lang_setting == 0) { cout << "Please enter the amount of 5,000 won bills
to be replenished." << endl; }
```

```
    replenish[1] = no_error(lang_setting);
    if (this->lang_setting == 1) { cout << "보충할 10000원권의 양을 입력해 주세요." << endl;
}
    if (this->lang_setting == 0) { cout << "Please enter the amount of 10,000 won bill
s to be replenished." << endl; }
    replenish[2] = no_error(lang_setting);
    if (this->lang_setting == 1) { cout << "보충할 50000원권의 양을 입력해 주세요." << endl;
}
    if (this->lang_setting == 0) { cout << "Please enter the amount of 50,000 won bill
s to be replenished." << endl; }
    replenish[3] = no_error(lang_setting);
    add_cash(replenish[0], replenish[1], replenish[2], replenish[3]);
    return;
  }
  else if (no == 0) {
    if (this->lang_setting == 1) { cout << "거래 내역을 출력합니다." << endl << endl; }
    else if (this->lang_setting == 0) { cout << "Printing transaction histories." << e
ndl << endl; }
    ifstream readFile(this->transaction_histories);
    if (readFile.is_open()) {
      string str;
      while (readFile) {
        getline(readFile, str);
        cout << str << endl;
      }
    }
    else {
      if (this->lang_setting == 1) { cout << "해당하는 파일이 없습니다." << endl; }
      if (this->lang_setting == 0) { cout << "No corresponding files exist." << endl;
}
    }
    return;
  }
  else {
    if (lang_setting == true) { cout << "초기 화면으로 돌아갑니다." << endl; }
    else { cout << "Returning to the initial screen." << endl; }
    return;
  }
  return;
}

void ATM::make_history(vector<string> rec) {
  //Transaction ID
  //Card Number
  //Transaction Types : deposit, withdraw, account transfer, cash_transfer
  //Amount
  //other transaction-specific information
  //account transfer:enemy account number
  //cash transfer:enemy account number
  //vector<string>new_history = { TransactionID, CardNumber, TransactionTypes, sorf, A
mount, Specific }; // TransactionID, CardNumber, TransactionTypes, Amount, Transaction
Specific Information
  //int len = static_cast<int>(rec.size());

  ofstream writeFromFile(this->transaction_histories, ios::app);
  writeFromFile << "TransactionID : " << rec[0] << "\n";
  writeFromFile << "CardNumber : " << rec[1] << "\n";
  writeFromFile << "TransactionTypes : " << rec[2] << "\n";
  writeFromFile << "Success or failure : " << rec[3] << "\n";
  writeFromFile << "Amount : " << rec[4] << "\n";
  writeFromFile << "Note : " << rec[5] << "\n";
  writeFromFile << "\n";
  return;
}

void ATM::display_transaction(vector<string> rec) {
  //string TransactionID, string CardNumber, string TransactionTypes, string sorf, str
ing Amount, string Specific
  cout << "Transaction ID : " << rec[0] << endl;
  cout << "CardNumber : " << rec[1] << endl;
  cout << "TransactionTypes : " << rec[2] << endl;
  cout << "Success or Failure : " << rec[3] << endl;
  cout << "Amount : " << rec[4] << endl;
  cout << "Note : " << rec[5] << endl;
  return;
}

void ATM::display_transaction_short(vector<string> rec) {
  //string TransactionID, string CardNumber, string TransactionTypes, string sorf, str
ing Amount, string Specific
  cout << "[/";
  for (int i = 0; i < rec.size(); i++) {
    if (rec[i] != "") { cout << rec[i] << "/"; }
  }
  cout << "]" << endl;
  return;
}

void ATM::printNow() {
  for (int i = 0; i < (*blist).size(); i++) {
    for (int j = 0; j < (*blist)[i]->get_account().size(); j++) {
      cout << "Account[" << (*blist)[i]->get_account()[j]->getUserName() << "] : " <<
(*blist)[i]->get_account()[j]->checkFunds() << "/" << (*blist)[i]->get_account()[j]->g
etAccountNumber() << "/" << (*blist)[i]->get_account()[j]->getBankName() << endl;
    }
  }
  for (int i = 0; i < (*alist).size(); i++) {
    cout << "ATM " << "[" << i << "] " << "Primary Bank : " << (*alist)[i]->getPrimary
()->getBankName() << " " << (*alist)[i]->getType() << " " << (*alist)[i]->getLangType
() << endl;
    cout << "Remaing cash : " << 1000 * (*alist)[i]->get1000() + 5000 * (*alist)[i]->g
et5000() + 10000 * (*alist)[i]->get10000() + 50000 * (*alist)[i]->get50000() << " (100
0 : " << (*alist)[i]->get1000() << ", 5000 : " << (*alist)[i]->get5000() << ", 10000 :
" << (*alist)[i]->get10000() << ", 50000 : " << (*alist)[i]->get50000() << ")" << end
l;
    // cout << "ATM [" << ATM_list[i]->getSerial() << "] Remaing cash : " << 1000 * AT
M_list[i]->get1000() + 5000 * ATM_list[i]->get5000() + 10000 * ATM_list[i]->get10000()
+ 50000 * ATM_list[i]->get50000() << " (1000 : " << ATM_list[i]->get1000() << ", 5000
: " << ATM_list[i]->get5000() << ", 10000 : " << ATM_list[i]->get10000() << ", 50000 :
" << ATM_list[i]->get50000() << ")" << endl;
  }
  cout << endl;
  cout << "Please re-enter here : ";
  return;
}

string ATM::getSerial() {
  return serial_number;
}

Bank* ATM::getPrimary() {
  return primary_bank;
}

string ATM::getLangType() {
  if (language_available == 1) {
    return "Unilingual";
  }
  else {
    return "Bilingual";
  }
}

int ATM::get1000() {
  return *(cash_storage[0]);
}
int ATM::get5000() {
  return *(cash_storage[1]);
}
int ATM::get10000() {
  return *(cash_storage[2]);
}
int ATM::get50000() {
  return *(cash_storage[3]);
}

//-------------------------------------------------Single Bank ATM Class Member Definiti
on
Single::Single(Bank* input_primary_bank, string input_serial_number, int input_lanuage
_available, int* initial_fund[], int* fees[4], vector<Bank*>* blist, vector<ATM*>* ali
st) : ATM(input_primary_bank, input_serial_number, 1, input_lanuage_available, initial
_fund, blist, alist) {
  for (int i = 0; i < 4; i++) {
    this->fee_list[i] = fees[i];
  }
}

int Single::deposit(Account* a) {
  if (this->lang_setting == true) {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌로 입금할 수 없습니다." << endl;
      return -1;
    }
    cout << "요금을 지불합니다." << endl << "수수료를 입금해 주세요." << endl;
    if (*(fee_list[0]) == 0) {
      cout << "지불할 수수료가 없습니다." << endl;
    }
    else {
      *(this->cash_storage[0]) += (*(fee_list[0])) / 1000;
      cout << "수수료가 입금되었습니다." << endl;
    }
    cout << "입금을 개시합니다." << endl;
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "Deposits cannot be made to accounts at other banks." << endl;
      return -1;
    }
    cout << "Pay the fee." << endl << "Please deposit the fee." << endl;
    if (*(fee_list[0]) == 0) {
      cout << "There are no fees to pay." << endl;
    }
    else {
      *(this->cash_storage[0]) += (*(fee_list[0])) / 1000;
      cout << "The fee has been deposited." << endl;
    }
    cout << "Initiate deposit." << endl;
  }
  int deposit_method;
  int cash1000;
  int cash5000;
  int cash10000;
  int cash50000;
  if (this->lang_setting == true) {
    cout << "입금 수단을 선택해 주세요" << endl;
    cout << "[1] 현금" << "   " << "[2] 수표" << endl;
    deposit_method = no_error_range(lang_setting, 1, 2);
    if (deposit_method == 1) {
      cout << "본 기기가 처리할 수 있는 지폐의 매수는 50장까지입니다." << endl << "1000원권을 투입해
주세요." << endl;
      cash1000 = no_error(lang_setting);
      cout << "5000원권을 투입해 주세요." << endl;
      cash5000 = no_error(lang_setting);
      cout << "10000원권을 투입해 주세요." << endl;
      cash10000 = no_error(lang_setting);
      cout << "50000원권을 투입해 주세요." << endl;
      cash50000 = no_error(lang_setting);
      if (cash1000 + cash5000 + cash10000 + cash50000 > 51) {
        cout << "기기의 처리 한계를 초과하였습니다." << endl;
        return -1;
      }
      add_cash(cash1000, cash5000, cash10000, cash50000);
      a->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50
000 * 50000));
      cout << "입금이 완료되었습니다." << endl << "입금 계좌의 잔고는 " << a->checkFunds() <<
"입니다." << endl;
      return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000
* 50000);
    }
    else if (deposit_method == 2) {
      cout << "본 기기가 처리할 수 있는 수표의 매수는 50장까지입니다. 수표를 투입해 주세요." << endl
<< "올바르지 않은 수표를 입력하면 투입이 중단됩니다." << endl;
      int checks[50];
      for (int i = 0; i <= 50; i++) {
        checks[i] = no_error(lang_setting);
        if (checks[i] < 100000) {
          checks[i] = 0;
          cout << "투입이 중단되었습니다. 입금을 시작합니다." << endl;
          break;
        }
      }
      int check_sum = 0;
      for (int i = 0; i <= 50; i++) {
        if (checks[i] >= 100000) {
          check_sum += checks[i];
        }
        else break;
      }
      a->deposit(check_sum);
      cout << "입금이 완료되었습니다." << endl << "입금 계좌의 잔고는 " << a->checkFunds() <<
"입니다." << endl;
      return check_sum;
    }
    else {
      cout << "오류가 발생했습니다. 입금을 중단합니다." << endl;
      return -1;
    }
  }
  else {
    cout << "Please select deposit method" << endl;
    cout << "[1] cash" << "    " << "[2] check" << endl;
    int deposit_method = no_error_range(lang_setting, 1, 2);
    if (deposit_method == 1) {
      cout << "This device can process up to 50 cashes." << endl << "Please input 1000
won bills." << endl;
      cash1000 = no_error(lang_setting);
      cout << "Please input 5000 won bills." << endl;
      cash5000 = no_error(lang_setting);
      cout << "Please input 10000 won bills." << endl;
      cash10000 = no_error(lang_setting);
      cout << "Please input 50000 won bills." << endl;
      cash50000 = no_error(lang_setting);
      if (cash1000 + cash5000 + cash10000 + cash50000 > 51) {
        cout << "The device's processing limit has been exceeded." << endl;
        return -1;
      }
      add_cash(cash1000, cash5000, cash10000, cash50000);
      a->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50
000 * 50000));
      cout << "Deposit has been completed." << endl << "The balance of the deposit acc
ount is " << setw(10) << a->checkFunds() << "." << endl;
      return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000
* 50000);
    }
    else if (deposit_method == 2) {
      cout << "This device can process up to 50 checks. Please insert checks." << endl
<< "If you enter an incorrect check, the insertion will be ended." << endl;
      int checks[50];
      for (int i = 0; i <= 50; i++) {
        checks[i] = no_error(lang_setting);
        if (checks[i] < 100000) {
          checks[i] = 0;
          cout << "Insertion ended. deposit is starting." << endl;
          break;
        }
      }
      int check_sum = 0;
      for (int i = 0; i <= 50; i++) {
        if (checks[i] >= 100000) {
          check_sum += checks[i];
        }
        else break;
      }
      a->deposit(check_sum);
      cout << "Deposit has been completed." << endl << "The balance of the deposit acc
ount is " << setw(10) << a->checkFunds() << "." << endl;
      return check_sum;
    }
    else {
      cout << "Error detected. Deposit will be stopped." << endl;
      return -1;
    }
  }
  return -1;
}

int Single::withdraw(Account* a) {
  if (this->lang_setting == true) {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌에서 출금할 수 없습니다." << endl;
      return -1;
    }
    cout << "요금을 지불합니다." << endl;
    if (*(this->fee_list[1]) == 0) {
      cout << "지불할 요금이 없습니다." << endl;
    }
    else if (*(this->fee_list[1]) > a->checkFunds()) {
      cout << "잔액이 부족합니다." << endl;
      return -1;
    }
    else {
      a->withdraw(*(this->fee_list[1]));
    }
    //this->withdraw(a);
    int amount;
    cout << "출금할 액수를 1000의 배수 단위로 입력하세요. 최대 금액은 50만원입니다." << endl;
    cout << "당신의 잔고는 " << setw(10) << a->checkFunds() << "원 입니다." << endl;
    amount = no_error(lang_setting);
    if (amount > 500000) {
      cout << "50만원을 초과한 금액을 입력하셨습니다. 출금을 취소합니다." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    if (amount % 1000 != 0) {
      cout << "1000의 배수가 아닌 금액을 입력하셨습니다. 출금을 취소합니다." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    if (amount > a->checkFunds()) {
      cout << "계좌에 잔액이 부족합니다. 출금을 취소합니다." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    int temp4 = amount / 50000;
    int temp3 = (amount % 50000) / 10000;
    int temp2 = (amount % 10000) / 5000;
    int temp1 = (amount % 5000) / 1000;
    if (temp4 > *(this->cash_storage[3])) {
      temp3 += 5 * (temp4 - *(this->cash_storage[3]));
    }
    if (temp3 > *(this->cash_storage[2])) {
      temp2 += 2 * (temp3 - *(this->cash_storage[2]));
    }
    if (temp2 > *(this->cash_storage[1])) {
      temp1 += 5 * (temp2 - *(this->cash_storage[1]));
    }
    if (temp1 > *(this->cash_storage[0])) {
      cout << "ATM에 현금이 부족합니다. 출금을 취소합니다." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    else {
      *(this->cash_storage[3]) -= temp4;
      *(this->cash_storage[2]) -= temp3;
      *(this->cash_storage[1]) -= temp2;
      *(this->cash_storage[0]) -= temp1;
    }
    a->withdraw(amount);
    cout << "출금이 완료되었습니다." << endl << "출금 계좌의 잔고는 " << setw(10) << a->checkFu
nds() << "입니다.";
    return amount;
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "Withdrawals cannot be made from accounts at other banks." << endl;
      return -1;
    }
    cout << "Pay the fee." << endl;
    if (*(this->fee_list[1]) == 0) {
      cout << "There are no fees to pay." << endl;
    }
    else if (*(this->fee_list[1]) > a->checkFunds()) {
      cout << "Your balance is insufficient." << endl;
      return -1;
    }
    else {
      a->withdraw(*(this->fee_list[1]));
    }
    //this->withdraw(a);
    int amount;
    cout << "Enter the amount you wish to withdraw in multiples of 1000. The maximum a
mount is 500,000 won." << endl;
```

```
    cout << "Your remaining fund is " << setw(10) << a->checkFunds() << "." << endl;
    amount = no_error(lang_setting);
    if (amount > 500000) {
      cout << "You entered an amount exceeding 500,000 won. Cancel withdrawal." << end
l;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    if (amount % 1000 != 0) {
      cout << "You entered an amount that is not a multiple of 1000. Cancel withdrawa
l." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    if (amount > a->checkFunds()) {
      cout << "There are insufficient funds in your account. Cancel withdrawal." << en
dl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    int temp4 = amount / 50000;
    int temp3 = (amount % 50000) / 10000;
    int temp2 = (amount % 10000) / 5000;
    int temp1 = (amount % 5000) / 1000;
    if (temp4 > *(this->cash_storage[3])) {
      temp3 += 5 * (temp4 - *(this->cash_storage[3]));
    }
    if (temp3 > *(this->cash_storage[2])) {
      temp2 += 2 * (temp3 - *(this->cash_storage[2]));
    }
    if (temp2 > *(this->cash_storage[1])) {
      temp1 += 5 * (temp2 - *(this->cash_storage[1]));
    }
    if (temp1 > *(this->cash_storage[0])) {
      cout << "The ATM is out of cash. Cancel withdrawal." << endl;
      a->deposit(*(this->fee_list[1]));
      return -1;
    }
    else {
      *(this->cash_storage[3]) -= temp4;
      *(this->cash_storage[2]) -= temp3;
      *(this->cash_storage[1]) -= temp2;
      *(this->cash_storage[0]) -= temp1;
    }
    a->withdraw(amount);
    cout << "Withdrawal has been completed." << endl << "The balance of the withdrawal
account is " << setw(10) << a->checkFunds() << ".";
    return amount;
  }
}

int Single::account_transfer(Account* a, Account* b) {
  if (this->lang_setting == false) {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "You cannot transfer money from an account at another bank." << endl;
      return -1;
    }
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "Money cannot be transferred to another bank's account." << endl;
      return -1;
    }
    cout << "Pay the fee." << endl;
    if (*(this->fee_list[2]) == 0) {
      cout << "There are no fees to pay." << endl;
    }
    else if (*(this->fee_list[2]) > a->checkFunds()) {
      cout << "Your balance is insufficient." << endl;
      return -1;
    }
    else {
      a->withdraw(*(this->fee_list[2]));
    }
    //this->account_transfer(a, b);
    int amount;
    cout << "Please enter the amount you wish to transfer." << endl;
    cout << "Your remaining fund is " << setw(10) << a->checkFunds() << "." << endl;
    amount = no_error(lang_setting);
    if (amount > a->checkFunds()) {
      cout << "Your balance is insufficient. Cancel the transfer." << endl;
      a->deposit(*(this->fee_list[2]));
      return -1;
    }
    a->withdraw(amount);
    b->deposit(amount);
    cout << "The transfer has been completed." << endl << "The balance of the source a
ccount is" << setw(10) << a->checkFunds() << ".";
    return amount;
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌에서 송금할 수 없습니다." << endl;
      return -1;
    }
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌로 송금할 수 없습니다." << endl;
      return -1;
    }
    cout << "요금을 지불합니다." << endl;
    if (*(this->fee_list[2]) == 0) {
      cout << "지불할 요금이 없습니다." << endl;
    }
    else if (*(this->fee_list[2]) > a->checkFunds()) {
      cout << "잔액이 부족합니다." << endl;
      return -1;
    }
    else {
      a->withdraw(*(this->fee_list[2]));
    }
    //this->account_transfer(a, b);
    int amount;
    cout << "송금할 액수를 입력해 주세요." << endl;
    cout << "당신의 잔고는 " << setw(10) << a->checkFunds() << "원 입니다." << endl;
    amount = no_error(lang_setting);
    if (amount > a->checkFunds()) {
      cout << "잔액이 부족합니다. 송금을 취소합니다." << endl;
      a->deposit(*(this->fee_list[2]));
      return -1;
    }
    a->withdraw(amount);
    b->deposit(amount);
    cout << "송금이 완료되었습니다." << endl << "출금 계좌의 잔고는 " << setw(10) << a->checkFu
nds() << "입니다.";
    return amount;
  }
}

int Single::cash_transfer(Account* b) {
  if (this->lang_setting == false) {
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "Money cannot be transferred to another bank's account." << endl;
      return -1;
    }
    cout << "Pay the fee." << endl << "Please deposit the fee." << endl;
    if (*(this->fee_list[3]) == 0) {
      cout << "There are no fees to pay." << endl;
    }
    else {
      *(this->cash_storage[0]) += (*(this->fee_list[3])) / 1000;
      cout << "The fee has been deposited." << endl;
    }
    cout << "Initiate cash transfer." << endl;
    cout << "Please insert a 1,000 won bill." << endl;
    int cash1000 = no_error(lang_setting);
    cout << "Please insert a 5,000 won bill." << endl;
    int cash5000 = no_error(lang_setting);
    cout << "Please insert a 10,000 won bill." << endl;
    int cash10000 = no_error(lang_setting);
    cout << "Please insert a 50,000 won bill." << endl;
    int cash50000 = no_error(lang_setting);
    add_cash(cash1000, cash5000, cash10000, cash50000);
    b->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash5000
0 * 50000));
    cout << "The transfer has been completed." << endl;
    return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000 *
50000);
  }
  else {
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌로 송금할 수 없습니다." << endl;
      return -1;
    }
    cout << "요금을 지불합니다." << endl << "수수료를 입금해 주세요." << endl;
    if (*(this->fee_list[3]) == 0) {
      cout << "지불할 수수료가 없습니다." << endl;
    }
    else {
      *(this->cash_storage[0]) += (*(this->fee_list[3])) / 1000;
      cout << "수수료가 입금되었습니다." << endl;
    }
    cout << "현금 송금을 개시합니다." << endl;
    cout << "1000원권을 투입해 주세요." << endl;
    int cash1000 = no_error(lang_setting);;
    cout << "5000원권을 투입해 주세요." << endl;
    int cash5000 = no_error(lang_setting);;
    cout << "10000원권을 투입해 주세요." << endl;
    int cash10000 = no_error(lang_setting);;
    cout << "50000원권을 투입해 주세요." << endl;
    int cash50000 = no_error(lang_setting);;
    add_cash(cash1000, cash5000, cash10000, cash50000);
    b->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash5000
0 * 50000));
    cout << "송금이 완료되었습니다." << endl;
    return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000 *
50000);
  }
}

Account* Single::card2account(string card, vector<Bank*> bank_list) {
  bool isPrimary = false;
  Account* ac = 0;
  for (int i = 0; i < primary_bank->get_account().size(); i++) {
    vector<string> card_list = primary_bank->get_account()[i]->getCardNumber();
    for (int j = 0; j < card_list.size(); j++) {
      if (card_list[j] == card) {
        isPrimary = true;
        ac = primary_bank->get_account()[i];
        break;
      }
    }
  }
  if (isPrimary == false) {
    if (this->lang_setting == 1) { cout << "지원되지 않는 카드입니다." << endl; }
    if (this->lang_setting == 0) { cout << "Unsupported card." << endl; }
    return nullptr;
  }
  return ac;
}

Account* Single::num2account(string num, vector<Bank*> bank_list) {
  bool isPrimary = false;
  Account* ac = 0;
  for (int i = 0; i < primary_bank->get_account().size(); i++) {
    string acc_num = primary_bank->get_account()[i]->getAccountNumber();
    if (acc_num == num) {
      isPrimary = true;
      ac = primary_bank->get_account()[i];
      break;
    }
  }
  if (isPrimary == false) {
    if (this->lang_setting == 1) { cout << "지원되지 않는 계좌입니다." << endl; }
    if (this->lang_setting == 0) { cout << "Unsupported account." << endl; }
    return nullptr;
  }
  return ac;
}

string Single::getType() {
  return "Single-bank ATM";
}

//-----------------------------------------------Multi Bank ATM Class Member Definiti
on
Multi::Multi(Bank* input_primary_bank, string input_serial_number, int input_lanuage_a
vailable, int* initial_fund[], int* fees[4], int* fees2[4], vector<Bank*>* blist, vect
or<ATM*>* alist) : ATM(input_primary_bank, input_serial_number, 2, input_lanuage_avail
able, initial_fund, blist, alist) {
  for (int i = 0; i < 4; i++) {
    this->fee_list[i] = fees[i];
  }
  for (int i = 0; i < 4; i++) {
    this->multi_fee_list[i] = fees2[i];
  }
}

int Multi::deposit(Account* a) {
  int account_count = 0;
  if (this->lang_setting == true) {
    if (a->getBankName() == this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌로 입금합니다." << endl;
      account_count = 1;
    }
    else {
      cout << "본 은행의 계좌로 입금합니다." << endl;
    }
    if (account_count == 0) {
      cout << "요금을 지불합니다." << endl << "수수료를 입금해 주세요." << endl;
      if (*(this->fee_list[0]) == 0) {
        cout << "지불할 수수료가 없습니다." << endl;
      }
      else {
        *(this->cash_storage[0]) += (*(this->fee_list[0])) / 1000;
        cout << "수수료가 입금되었습니다." << endl;
      }
      cout << "입금을 개시합니다." << endl;
    }
    else {
      cout << "요금을 지불합니다." << endl << "수수료를 입금해 주세요." << endl;
      if (*(this->multi_fee_list[0]) == 0) {
        cout << "지불할 수수료가 없습니다." << endl;
      }
      else {
        *(this->cash_storage[0]) += (*(this->multi_fee_list[0])) / 1000;
        cout << "수수료가 입금되었습니다." << endl;
      }
      cout << "입금을 개시합니다." << endl;
    }
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "Deposits will be made to accounts at other banks." << endl;
      account_count = 1;
    }
    if (account_count == 0) {
      cout << "Pay the fee." << endl << "Please deposit the fee." << endl;
      if (*(this->fee_list[0]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else {
        *(this->cash_storage[0]) += (*(this->fee_list[0])) / 1000;
        cout << "The fee has been deposited." << endl;
      }
      cout << "Initiate deposit." << endl;
    }
    else {
      cout << "Pay the fee." << endl << "Please deposit the fee." << endl;
      if (*(this->multi_fee_list[0]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else {
        *(this->cash_storage[0]) += (*(this->multi_fee_list[0])) / 1000;
        cout << "The fee has been deposited." << endl;
      }
      cout << "Initiate deposit." << endl;
    }
  }
  int deposit_method;
  if (this->lang_setting == true) {
    cout << "입금 수단을 선택해 주세요" << endl;
    cout << "[1] 현금" << "    " << "[2] 수표" << endl;
    deposit_method = no_error_range(lang_setting, 1, 2);
    if (deposit_method == 1) {
      cout << "본 기기가 처리할 수 있는 지폐의 매수는 50장까지입니다." << endl << "1000원권을 투입해
주세요." << endl;
      int cash1000 = no_error(lang_setting);
      cout << "5000원권을 투입해 주세요." << endl;
      int cash5000 = no_error(lang_setting);
      cout << "10000원권을 투입해 주세요." << endl;
      int cash10000 = no_error(lang_setting);
      cout << "50000원권을 투입해 주세요." << endl;
      int cash50000 = no_error(lang_setting);
      if (cash1000 + cash5000 + cash10000 + cash50000 > 51) {
        cout << "기기의 처리 한계를 초과하였습니다." << endl;
        return -1;
      }
      add_cash(cash1000, cash5000, cash10000, cash50000);
      a->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50
000 * 50000));
      cout << "입금이 완료되었습니다." << endl << "입금 계좌의 잔고는 " << setw(10) << a->check
Funds() << "입니다." << endl;
      return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000
* 50000);
    }
    else if (deposit_method == 2) {
      cout << "본 기기가 처리할 수 있는 수표의 매수는 50장까지입니다. 수표를 투입해 주세요." << endl
<< "올바르지 않은 수표를 입력하면 투입이 중단됩니다." << endl;
```

```
      int checks[50];
      for (int i = 0; i <= 50; i++) {
        checks[i] = no_error(lang_setting);
        if (checks[i] < 100000) {
          checks[i] = 0;
          cout << "투입이 중단되었습니다. 입금을 시작합니다." << endl;
          break;
        }
      }
      int check_sum = 0;
      for (int i = 0; i <= 50; i++) {
        if (checks[i] >= 100000) {
          check_sum += checks[i];
        }
        else break;
      }
      a->deposit(check_sum);
      cout << "입금이 완료되었습니다." << endl << "입금 계좌의 잔고는 " << setw(10) << a->check
Funds() << "입니다." << endl;
      return check_sum;
    }
    else {
      cout << "오류가 발생했습니다. 입금을 중단합니다." << endl;
      return -1;
    }
  }
  else {
    cout << "Please select deposit method" << endl;
    cout << "[1] cash" << "    " << "[2] check" << endl;
    int deposit_method = no_error_range(lang_setting, 1, 2);
    if (deposit_method == 1) {
      cout << "This device can process up to 50 cashes." << endl << "Please input 1000
won bills." << endl;
      int cash1000 = no_error(lang_setting);
      cout << "Please input 5000 won bills." << endl;
      int cash5000 = no_error(lang_setting);
      cout << "Please input 10000 won bills." << endl;
      int cash10000 = no_error(lang_setting);
      cout << "Please input 50000 won bills." << endl;
      int cash50000 = no_error(lang_setting);
      if (cash1000 + cash5000 + cash10000 + cash50000 > 51) {
        cout << "The device's processing limit has been exceeded." << endl;
        return -1;
      }
      add_cash(cash1000, cash5000, cash10000, cash50000);
      a->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50
000 * 50000));
      cout << "Deposit has been completed." << endl << "The balance of the deposit acc
ount is " << setw(10) << a->checkFunds() << "." << endl;
      return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000
* 50000);
    }
    else if (deposit_method == 2) {
      cout << "This device can process up to 50 checks. Please insert checks." << endl
<< "If you enter an incorrect check, the insertion will be ended." << endl;
      int checks[50];
      for (int i = 0; i <= 50; i++) {
        checks[i] = no_error(lang_setting);
        if (checks[i] < 100000) {
          checks[i] = 0;
          cout << "Insertion ended. deposit is starting." << endl;
          break;
        }
      }
      int check_sum = 0;
      for (int i = 0; i <= 50; i++) {
        if (checks[i] >= 100000) {
          check_sum += checks[i];
        }
        else break;
      }
      a->deposit(check_sum);
      cout << "Deposit has been completed." << endl << "The balance of the deposit acc
ount is " << setw(10) << a->checkFunds() << "." << endl;
      return check_sum;
    }
    else {
      cout << "Error detected. Deposit will be stopped." << endl;
      return -1;
    }
  }
  return -1;
}

int Multi::withdraw(Account* a) {
  int coconut = 0;
  if (this->lang_setting == true) {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌에서 출금합니다." << endl;
      coconut = 1;
    }
    else {
      cout << "본 은행의 계좌에서 출금합니다." << endl;
    }
    cout << "요금을 지불합니다." << endl;
    if (coconut == 0) {
      if (*(this->fee_list[1]) == 0) {
        cout << "지불할 요금이 없습니다." << endl;
      }
      else if (*(this->fee_list[1]) > a->checkFunds()) {
        cout << "잔액이 부족합니다." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->fee_list[1]));
      }
    }
    else {
      if (*(this->multi_fee_list[1]) == 0) {
        cout << "지불할 요금이 없습니다." << endl;
      }
      else if (*(this->multi_fee_list[1]) > a->checkFunds()) {
        cout << "잔액이 부족합니다." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[1]));
      }
    }
    //this->withdraw(a);
    cout << "출금할 액수를 1000의 배수 단위로 입력하세요. 최대 금액은 50만원입니다." << endl;
    cout << "당신의 잔고는 " << setw(10) << a->checkFunds() << "원 입니다." << endl;
    int amount = no_error(lang_setting);
    if (amount > 500000) {
      cout << "50만원을 초과한 금액을 입력하셨습니다. 출금을 취소합니다." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    if (amount % 1000 != 0) {
      cout << "1000의 배수가 아닌 금액을 입력하셨습니다. 출금을 취소합니다." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    if (amount > a->checkFunds()) {
      cout << "계좌에 잔액이 부족합니다. 출금을 취소합니다." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    int temp4 = amount / 50000;
    int temp3 = (amount % 50000) / 10000;
    int temp2 = (amount % 10000) / 5000;
    int temp1 = (amount % 5000) / 1000;
    if (temp4 > *(this->cash_storage[3])) {
      temp3 += 5 * (temp4 - *(this->cash_storage[3]));
    }
    if (temp3 > *(this->cash_storage[2])) {
      temp2 += 2 * (temp3 - *(this->cash_storage[2]));
    }
    if (temp2 > *(this->cash_storage[1])) {
      temp1 += 5 * (temp2 - *(this->cash_storage[1]));
    }
    if (temp1 > *(this->cash_storage[0])) {
      cout << "ATM에 현금이 부족합니다. 출금을 취소합니다." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    else {
      *(this->cash_storage[3]) -= temp4;
      *(this->cash_storage[2]) -= temp3;
      *(this->cash_storage[1]) -= temp2;
      *(this->cash_storage[0]) -= temp1;
    }
    a->withdraw(amount);
    cout << "출금이 완료되었습니다." << endl << "출금 계좌의 잔고는 " << setw(10) << a->checkFu
nds() << "입니다.";
    return amount;
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "Withdrawals will be made from accounts at other banks." << endl;
      coconut = 1;
    }
    else {
      cout << "Withdrawals will be made from accounts at this banks." << endl;
    }
    if (coconut == 0) {
      cout << "Pay the fee." << endl;
      if (*(this->fee_list[1]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else if (*(this->fee_list[1]) > a->checkFunds()) {
        cout << "Your balance is insufficient." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->fee_list[1]));
      }
    }
    else {
      cout << "Pay the fee." << endl;
      if (*(this->multi_fee_list[1]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else if (*(this->multi_fee_list[1]) > a->checkFunds()) {
        cout << "Your balance is insufficient." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[1]));
      }
    }
    //this->withdraw(a);
    cout << "Enter the amount you wish to withdraw in multiples of 1000. The maximum a
mount is 500,000 won." << endl;
```

```
    cout << "Your remaining fund is " << setw(10) << a->checkFunds() << "." << endl;
    int amount = no_error(lang_setting);
    if (amount > 500000) {
      cout << "You entered an amount exceeding 500,000 won. Cancel withdrawal." << end
l;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    if (amount % 1000 != 0) {
      cout << "You entered an amount that is not a multiple of 1000. Cancel withdrawa
l." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    if (amount > a->checkFunds()) {
      cout << "There are insufficient funds in your account. Cancel withdrawal." << en
dl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    int temp4 = amount / 50000;
    int temp3 = (amount % 50000) / 10000;
    int temp2 = (amount % 10000) / 5000;
    int temp1 = (amount % 5000) / 1000;
    if (temp4 > *(this->cash_storage[3])) {
      temp3 += 5 * (temp4 - *(this->cash_storage[3]));
    }
    if (temp3 > *(this->cash_storage[2])) {
      temp2 += 2 * (temp3 - *(this->cash_storage[2]));
    }
    if (temp2 > *(this->cash_storage[1])) {
      temp1 += 5 * (temp2 - *(this->cash_storage[1]));
    }
    if (temp1 > *(this->cash_storage[0])) {
      cout << "The ATM is out of cash. Cancel withdrawal." << endl;
      if (coconut == 0) {
        a->deposit(*(this->fee_list[1]));
      }
      else {
        a->deposit(*(this->multi_fee_list[1]));
      }
      return -1;
    }
    else {
      *(this->cash_storage[3]) -= temp4;
      *(this->cash_storage[2]) -= temp3;
      *(this->cash_storage[1]) -= temp2;
      *(this->cash_storage[0]) -= temp1;
    }
    a->withdraw(amount);
    cout << "Withdrawal has been completed." << endl << "The balance of the withdrawal
account is " << setw(10) << a->checkFunds() << ".";
    return amount;
  }
}

int Multi::account_transfer(Account* a, Account* b) {
  int coconut = 0;
  int cococonut = 0;
  if (this->lang_setting == false) {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "Transfer money from an account at another bank : " << a->getUserName()
<< endl;
      coconut = 1;
    }
    //else {
    if (a->getBankName() == this->primary_bank->getBankName()) {
      cout << "Transfer money from an account at this bank : " << a->getUserName() <<
endl;
    }
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "Transferred to another bank's account : " << b->getUserName() << endl;
      cococonut = 1;
    }
    //else {
    if (b->getBankName() == this->primary_bank->getBankName()) {
      cout << "Transferred to this bank's account : " << b->getUserName() << endl;
    }
    if (coconut == 0 && cococonut == 0) {
      cout << "Pay the fee." << endl;
      if (*(this->fee_list[2]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else if (*(this->fee_list[2]) > a->checkFunds()) {
        cout << "Your balance is insufficient." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->fee_list[2]));
      }
    }
    else if (coconut == 1 && cococonut == 1) {
      cout << "Pay the fee." << endl;
      if (*(this->multi_fee_list[3]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else if (*(this->multi_fee_list[3]) > a->checkFunds()) {
        cout << "Your balance is insufficient." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[3]));
      }
    }
    else {
      cout << "Pay the fee." << endl;
      if (*(this->multi_fee_list[2]) == 0) {
        cout << "There are no fees to pay." << endl;
      }
      else if (*(this->multi_fee_list[2]) > a->checkFunds()) {
        cout << "Your balance is insufficient." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[2]));
      }
    }
    //this->account_transfer(a, b);
    cout << "Please enter the amount you wish to transfer." << endl;
    cout << "Your remaining fund is " << setw(10) << a->checkFunds() << "." << endl;
    int amount = no_error(lang_setting);
    if (a->checkFunds() < amount) {
      cout << "Your balance is insufficient. Cancel the transfer." << endl;
      if (coconut == 0 && cococonut == 0) {
        a->deposit(*(this->fee_list[2]));
      }
      else if (coconut == 1 && cococonut == 1) {
        a->deposit(*(this->multi_fee_list[3]));
      }
      else {
        a->deposit(*(this->multi_fee_list[2]));
      }
      return -1;
    }
    a->withdraw(amount);
    b->deposit(amount);
    cout << "The transfer has been completed." << endl << "The balance of the source a
ccount is" << setw(10) << a->checkFunds() << "." << endl;
    return amount;
  }
  else {
    if (a->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌에서 송금합니다 : " << a->getUserName() << endl;
      coconut = 1;
    }
    else {
      cout << "본 은행의 계좌에서 송금합니다 : " << a->getUserName() << endl;
    }
    if (b->getBankName() != this->primary_bank->getBankName()) {
      cout << "타 은행의 계좌로 송금합니다 : " << b->getUserName() << endl;
      cococonut = 1;
    }
    else {
      cout << "본 은행의 계좌로 송금합니다 : " << b->getUserName() << endl;
    }
    cout << "요금을 지불합니다." << endl;
    if (coconut == 0 && cococonut == 0) {
      if (*(this->fee_list[2]) == 0) {
        cout << "지불할 요금이 없습니다." << endl;
      }
      else if (*(this->fee_list[2]) > a->checkFunds()) {
        cout << "잔액이 부족합니다." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->fee_list[2]));
      }
    }
    else if (coconut == 1 && cococonut == 1) {
      if (*(this->multi_fee_list[3]) == 0) {
        cout << "지불할 요금이 없습니다." << endl;
      }
      else if (*(this->multi_fee_list[3]) > a->checkFunds()) {
        cout << "잔액이 부족합니다." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[3]));
      }
    }
    else {
      if (*(this->multi_fee_list[2]) == 0) {
        cout << "지불할 요금이 없습니다." << endl;
      }
      else if (*(this->multi_fee_list[2]) > a->checkFunds()) {
        cout << "잔액이 부족합니다." << endl;
        return -1;
      }
      else {
        a->withdraw(*(this->multi_fee_list[2]));
      }
    }
    //this->account_transfer(a, b);
    cout << "송금할 액수를 입력해 주세요." << endl;
    cout << "보유하신 잔고는 " << setw(10) << a->checkFunds() << "원 입니다." << endl;
    int amount = no_error(lang_setting);
    if (amount > a->checkFunds()) {
      cout << "잔액이 부족합니다. 송금을 취소합니다." << endl;
      if (coconut == 0 && cococonut == 0) {
        a->deposit(*(this->fee_list[2]));
      }
      else if (coconut == 0 && cococonut == 0) {
        a->deposit(*(this->multi_fee_list[3]));
      }
      else {
        a->deposit(*(this->multi_fee_list[2]));
      }
      return -1;
    }
```

```
    a->withdraw(amount);
    b->deposit(amount);
    cout << "송금이 완료되었습니다." << endl << "출금 계좌의 잔고는 " << setw(10) << a->checkFu
nds() << "입니다." << endl;
    return amount;
  }
}

int Multi::cash_transfer(Account* b) {
  if (this->lang_setting == false) {
    cout << "Pay the fee." << endl << "Please deposit the fee." << endl;
    *(this->cash_storage[0]) += (*(this->fee_list[3])) / 1000;
    cout << "The fee has been deposited. Initiate cash transfer." << endl;
    //this->cash_transfer(b);
    cout << "Please insert a 1,000 won bill." << endl;
    int cash1000 = no_error(lang_setting);
    cout << "Please insert a 5,000 won bill." << endl;
    int cash5000 = no_error(lang_setting);
    cout << "Please insert a 10,000 won bill." << endl;
    int cash10000 = no_error(lang_setting);
    cout << "Please insert a 50,000 won bill." << endl;
    int cash50000 = no_error(lang_setting);
    add_cash(cash1000, cash5000, cash10000, cash50000);
    b->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash5000
0 * 50000));
    cout << "The transfer has been completed." << endl;
    return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000 *
50000);
  }
  else {
    cout << "요금을 지불합니다." << endl << "수수료를 입금해 주세요." << endl;
    *(this->cash_storage[0]) += (*(this->fee_list[3])) / 1000;
    cout << "수수료가 입금되었습니다. 현금 송금을 개시합니다." << endl;
    //this->cash_transfer(b);
    cout << "1000원권을 투입해 주세요." << endl;
    int cash1000 = no_error(lang_setting);
    cout << "5000원권을 투입해 주세요." << endl;
    int cash5000 = no_error(lang_setting);
    cout << "10000원권을 투입해 주세요." << endl;
    int cash10000 = no_error(lang_setting);
    cout << "50000원권을 투입해 주세요." << endl;
    int cash50000 = no_error(lang_setting);
    add_cash(cash1000, cash5000, cash10000, cash50000);
    b->deposit((cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash5000
0 * 50000));
    cout << "송금이 완료되었습니다." << endl;
    return (cash1000 * 1000) + (cash5000 * 5000) + (cash10000 * 10000) + (cash50000 *
50000);
  }
}

Account* Multi::card2account(string card, vector<Bank*> bank_list) {
  int* banknum = new int(-1);
  Account* ac = 0;
  for (int k = 0; k < bank_list.size(); k++) {
    for (int i = 0; i < bank_list[k]->get_account().size(); i++) {
      vector<string> card_list = bank_list[k]->get_account()[i]->getCardNumber();
      for (int j = 0; j < card_list.size(); j++) {
        if (card_list[j] == card) {
          *banknum = k;
          return bank_list[k]->get_account()[i];
          delete banknum;
          break;
        }
      }
    }
  }
  if (*banknum == -1) {
    if (this->lang_setting == true) { cout << "지원되지 않는 카드입니다." << endl; }
    else { cout << "Unsupported card." << endl; }
    delete banknum;
    return nullptr;
  }
  delete banknum;
  return ac;
}

Account* Multi::num2account(string num, vector<Bank*> bank_list) {
  int* banknum = new int(-1);
  Account* ac = nullptr;
  for (int k = 0; k < bank_list.size(); k++) {
    for (int i = 0; i < bank_list[k]->get_account().size(); i++) {
      string acc_num = bank_list[k]->get_account()[i]->getAccountNumber();
      if (acc_num == num) {
        *banknum = k;
        ac = bank_list[*banknum]->get_account()[i];
        break;
      }
    }
  }
  //delete banknum;
  return ac;
}

string Multi::getType() {
  return "Multi-bank ATM";
}

//----------------------------------------------------------General Function Definiti
on
void printNow(vector<ATM*>& ATM_list, vector<Bank*>& bank_list) {

  for (int i = 0; i < bank_list.size(); i++) {
    for (int j = 0; j < bank_list[i]->get_account().size(); j++) {
      cout << "Account[" << (bank_list)[i]->get_account()[j]->getUserName() << "] : "
<< (bank_list)[i]->get_account()[j]->checkFunds() << "/" << (bank_list)[i]->get_accoun
t()[j]->getAccountNumber() << "/" << (bank_list)[i]->get_account()[j]->getBankName() <
< endl;
    }
  }
  for (int i = 0; i < (ATM_list).size(); i++) {
    cout << "ATM " << "[" << i << "] " << "Primary Bank : " << (ATM_list)[i]->getPrima
```

```
ry()->getBankName() << " " << (ATM_list)[i]->getType() << " " << (ATM_list)[i]->getLan
gType() << endl;
    cout << "Remaing cash : " << 1000 * (ATM_list)[i]->get1000() + 5000 * (ATM_list)
[i]->get5000() + 10000 * (ATM_list)[i]->get10000() + 50000 * (ATM_list)[i]->get50000()
<< " (1000 : " << (ATM_list)[i]->get1000() << ", 5000 : " << (ATM_list)[i]->get5000()
<< ", 10000 : " << (ATM_list)[i]->get10000() << ", 50000 : " << (ATM_list)[i]->get5000
0() << ")" << endl;
  }

  cout << endl;
  cout << "Please re-enter here : ";
  return;
}

string Qsearch(vector<ATM*>& ATM_list, vector<Bank*>& bank_list) {
  string str;
  while (true) {
    cin >> str;
    if (str == "Q" || str == "q") {
      printNow(ATM_list, bank_list);
    }
    else {
      return str;
    }
  }
}

int no_error(vector<ATM*>& ATM_list, vector<Bank*>& bank_list, int language_setting) {
  int temp = 0;
  while (true) {

    string abc = "";
    // cin >> abc;
    while (true) {
      cin >> abc;
      if (abc == "Q" || abc == "q") {
        printNow(ATM_list, bank_list);
      }
      else {
        break;
      }
    }

    if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
("\n") != string::npos) {
      if (language_setting == 1) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
    if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
      temp = stoi(abc);
      if (temp >= 0) { return temp; }
      else {
        if (language_setting == 1) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 2) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
    if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
      if (language_setting == 1) {
        cout << "[Error] Input out of range. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
      continue;
    }
  }
}

int no_error_range(vector<ATM*>& ATM_list, vector<Bank*>& bank_list, int language_sett
ing, int min, int max) {
  int temp = 0;
  while (true) {

    string abc = "";
    // cin >> abc;
    while (true) {
      cin >> abc;
      if (abc == "Q" || abc == "q") {
        printNow(ATM_list, bank_list);
      }
      else {
        break;
      }
    }

    if (abc.find(".") != string::npos || abc.find("-") != string::npos || abc.find
("\n") != string::npos) {
      if (language_setting == 1) {
        cout << "[Error] An input error has occurred. Please write again." << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "[에러] 입력 오류가 발생했습니다. 다시 한 번 입력해 주세요." << endl;
        cout << "숫자를 입력해주세요 : ";
      }
```

```
        continue;
      }

      if (atoi(abc.c_str()) != 0 || abc.compare("0") == 0) {
        temp = stoi(abc);
        if (temp > min - 1 && temp < max + 1) { return temp; }
        else {
          if (language_setting == 1) {
            cout << "[Error] Input out of range. Please write again." << endl;
            cout << "Please Enter the Number : ";
          }
          if (language_setting == 2) {
            cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
            cout << "숫자를 입력해주세요 : ";
          }
          continue;
        }
      }
      if (atoi(abc.c_str()) == 0 && abc.compare("0") != 0) {
        if (language_setting == 1) {
          cout << "[Error] Input out of range. Please write again." << endl;
          cout << "Please Enter the Number : ";
        }
        if (language_setting == 2) {
          cout << "[에러] 범위 밖의 입력. 다시 한 번 입력해 주세요." << endl;
          cout << "숫자를 입력해주세요 : ";
        }
        continue;
      }
    }
  }

  void BankMake(vector<ATM*>& ATM_list, vector<Bank*>& bank_list, int language_setting,
  vector<Bank*>* blist, vector<ATM*>* alist) {

    cout << "==================== < Bank Duplicate Session > ====================" << en
  dl;
    string bank_name; string bank_name_save;

    while (true) {
      int dup = 0;

      if (language_setting == 1) { cout << "Please write bank name : "; }
      if (language_setting == 2) { cout << "은행명을 입력해주세요. : "; }

      cin >> bank_name;
      // Qsearch(ATM_list, bank_list, bank_name);

      bank_name_save = bank_name;

      for (int i = 0; i < bank_name.size(); i++) {
        bank_name[i] = tolower(bank_name[i]);
      }

      for (int j = 0; j < bank_list.size(); j++) {
```

```
      // string reference = bank_list[j].getBankName();
      string reference = bank_list[j]->getBankName();

      for (int k = 0; k < reference.size(); k++) {
        reference[k] = tolower(reference[k]);
      }
      if (reference == bank_name) {
        if (language_setting == 1) { cout << bank_name << " is already exiested!" << e
ndl; }
        if (language_setting == 2) { cout << bank_name << " 은행은 이미 존재합니다!" << end
l; }
        dup = 1;
      }
    }
    if (dup == 0) {
      // bank_list.push_back(Bank(bank_name_save));
      bank_list.push_back(new Bank(bank_name_save, blist, alist));
      cout << "==================== < Bank Duplicate Session End! > ==================
==" << endl;
      break;
    }
  }
}

void BankService(vector<ATM*>& ATM_list, vector<Bank*> bank_list, int language_settin
g) {

  cout << "==================== < Bank Service Session > ====================" << end
l;

  if (bank_list.size() == 0) {
    if (language_setting == 1) { cout << "There is no bank. So bank service is not ava
ilable." << endl; }
    if (language_setting == 2) { cout << "은행이 없어 은행 서비스를 수행할 수 없습니다." << end
l; }
    return;
  }

  if (language_setting == 1) { cout << "Please choose Bank number that you want to mak
e account for." << endl; }
  if (language_setting == 2) { cout << "계좌를 만들고 싶은 은행을 선택해주세요." << endl; }

  for (int i = 0; i < bank_list.size(); i++) {
    cout << "[" << i << "] " << bank_list[i]->getBankName() << " Bank" << endl;
  }
  cout << endl;

  if (language_setting == 1) { cout << "Bank number : "; }
  if (language_setting == 2) { cout << "은행 번호 : "; }

  int bank_choose = 0;
  bank_choose = no_error_range(ATM_list, bank_list, language_setting, 0, (int)bank_lis
t.size() - 1);

  if (language_setting == 1) { cout << bank_list[bank_choose]->getBankName() << " bank
is selected." << endl; }
```

```
  if (language_setting == 2) { cout << bank_list[bank_choose]->getBankName() << " 은행이 
선택되었습니다." << endl; }

  if (language_setting == 1) {
    cout << "Please choose number that you want to get service." << endl;
    cout << "[1] Create Account" << endl;
    cout << "[2] Make Card" << endl;
    cout << "[3] Account history" << endl;
    cout << endl;
    cout << "Serive number : ";
  }
  if (language_setting == 2) {
    cout << "이용하고자 하는 서비스를 선택해주세요." << endl;
    cout << "[1] 계좌 개설" << endl;
    cout << "[2] 카드 생성" << endl;
    cout << "[3] 계좌 기록" << endl;
    cout << endl;
    cout << "서비스 번호 : ";
  }

  int service_choose = 0;
  // cin >> service_choose;
  service_choose = no_error_range(ATM_list, bank_list, language_setting, 1, 3);


  if (service_choose == 1) {
    bank_list[bank_choose]->create_account(language_setting);
  }
  if (service_choose == 2) {
    bank_list[bank_choose]->makeCard_session(language_setting);
  }
  if (service_choose == 3) {
    Account* input_account = bank_list[bank_choose]->search_account_number(language_se
tting);
    if (input_account == NULL) { return; }
    string input_password;
    int a = 3;
    while (true) {

      if (a == 0) {
        if (language_setting == 1) { cout << "You write wrong password 3 times. " << e
ndl; }
        if (language_setting == 2) { cout << "비밀번호 3회 틀렸습니다." << endl; }
        cout << "=================== < Account History Session End! > ===============
=====" << endl;
        return;
      }
      if (language_setting == 1) {
        cout << a << " attempts left." << endl;
        cout << "Password : ";
      }
      if (language_setting == 2) {
        cout << a << " 회 남음." << endl;
        cout << "비밀번호 : ";
      }
      // password is string!
      a--;
      //cin >> input_password;
      input_password = Qsearch(ATM_list, bank_list);
      //cout << input_password;

      if (input_account->getPassword() == input_password) {
        if (language_setting == 1) { cout << "Correct password." << endl; }
        if (language_setting == 2) { cout << "옳은 비밀번호." << endl; }
        cout << endl;
        input_account->printHistory();
        cout << "==================== < Account History Session End! > ===============
=====" << endl;
        return;
      }
    }
  }

  cout << "==================== < Bank Service Session End! > ====================" <<
endl;
}

void ATMMake(vector<ATM*>& ATM_list, vector<Bank*>& bank_list, int* fee_list1[4], int*
fee_list2[4], int language_setting, vector<Bank*>* blist, vector<ATM*>* alist) {
  cout << "==================== < ATM Duplicate Session > ====================" << end
l;
  //input : primary bank name, serial number, type, language, initial fund
  //constant :
  if (language_setting == 1) { cout << "Please choose Bank number that you want to mak
e ATM for : " << endl; }
  if (language_setting == 2) { cout << "ATM을 설치할 은행을 선택하세요 : " << endl; }
  for (int i = 0; i < bank_list.size(); i++) {
    cout << "[" << i << "] " << bank_list[i]->getBankName() << " Bank" << endl;
  }

  int bank_choose = 0;
  // cin >> bank_choose;
  bank_choose = no_error_range(ATM_list, bank_list, language_setting, 0, (int)bank_lis
t.size() - 1);

  if (bank_choose >= bank_list.size()) {
    if (language_setting == 1) { cout << "The corresponding bank does not exist." << e
ndl; }
    if (language_setting == 2) { cout << "해당하는 은행이 존재하지 않습니다." << endl; }
    cout << "==================== < ATM Make Session End! > ====================" << e
ndl;
    return;
  }
  if (language_setting == 1) { cout << bank_list[bank_choose]->getBankName() << " bank
is selected." << endl; }
  if (language_setting == 2) { cout << bank_list[bank_choose]->getBankName() << " 은행이
선택되었습니다.." << endl; }

  string serial = "";
  for (int i = 0; i < 6 - to_string(ATM_list.size()).size(); i++) {
    serial += "0";
  }
  serial += to_string(ATM_list.size());
  if (language_setting == 1) { cout << "Serial number of ATM will be automatically iss
ued. Serial number of this ATM is " << serial << "." << endl; }
  if (language_setting == 2) { cout << "ATM의 일련번호가 자동으로 발급됩니다. 본 ATM의 일련번호는
" << serial << "입니다." << endl; }

  if (language_setting == 1) { cout << "Please input ATM type." << endl << "[1] Single
Bank ATM" << endl << "[2] Multi Bank ATM" << endl; }
  if (language_setting == 2) { cout << "ATM의 type을 입력하세요." << endl << "[1] Single B
ank ATM" << endl << "[2] Multi Bank ATM" << endl; }

  int type = 0;
  // cin >> type;
  type = no_error_range(ATM_list, bank_list, language_setting, 1, 2);

  if (language_setting == 1) { cout << "Please input ATM language type." << endl << "
[1] Unilingual ATM" << endl << "[2] Bilingual ATM" << endl; }
  if (language_setting == 2) { cout << "ATM의 언어 type을 선택해 주세요." << endl << "[1] U
nilingual ATM" << endl << "[2] Bilingual ATM" << endl; }

  int language = 0;
  // cin >> language;
  language = no_error_range(ATM_list, bank_list, language_setting, 1, 2);

  int* fund[4] = { 0, };
  for (int i = 0; i < 4; i++) {
    fund[i] = new int(0);
  }
  if (language_setting == 1) { cout << "Please Enter the Amount of 1000 won bills." <<
endl; }
  if (language_setting == 2) { cout << "1000원권의 수를 입력해 주세요." << endl; }
  // cin >> *(fund[0]);
  *(fund[0]) = no_error(ATM_list, bank_list, language_setting);

  if (language_setting == 1) { cout << "Please Enter the Amount of 5000 won bills." <<
endl; }
  if (language_setting == 2) { cout << "5000원권의 수를 입력해 주세요." << endl; }
  // cin >> *(fund[1]);
  *(fund[1]) = no_error(ATM_list, bank_list, language_setting);

  if (language_setting == 1) { cout << "Please Enter the Amount of 10000 won bills." <
< endl; }
  if (language_setting == 2) { cout << "10000원권의 수를 입력해 주세요." << endl; }
  // cin >> *(fund[2]);
  *(fund[2]) = no_error(ATM_list, bank_list, language_setting);

  if (language_setting == 1) { cout << "Please Enter the Amount of 50000 won bills." <
< endl; }
  if (language_setting == 2) { cout << "50000원권의 수를 입력해 주세요." << endl; }
  // cin >> *(fund[3]);
  *(fund[3]) = no_error(ATM_list, bank_list, language_setting);

  switch (type) {
  case 1:
    //Single(bank_list[bank_choose].getBankName(), serial, language, fund, fee_list1);
    ATM_list.push_back(new Single(bank_list[bank_choose], serial, language, fund, fee_
list1, blist, alist));
    cout << "==================== < ATM Duplicate Session End! > ===================="
<< endl;
    break;
  case 2:
    //Multi(bank_list[bank_choose].getBankName(), serial, language, fund, fee_list1, f
ee_list2);
    ATM_list.push_back(new Multi(bank_list[bank_choose], serial, language, fund, fee_l
ist1, fee_list2, blist, alist));
    cout << "==================== < ATM Duplicate Session End! > ===================="
<< endl;
    break;
  default:
    break;
  }

  return;
}

void ATMService(vector<ATM*> ATM_list, vector<Bank*> bank_list, int language_setting)
{
  cout << "==================== < ATM Service Session > ====================" << endl;

  if (ATM_list.size() == 0) {
    if (language_setting == 1) { cout << "There is no ATM existed." << endl; }
    if (language_setting == 2) { cout << "존재하는 ATM이 없습니다." << endl; }
    return;
  }

  if (language_setting == 1) { cout << "Please choose ATM number that you want to use
for." << endl; }
  if (language_setting == 2) { cout << "거래를 진행하고 싶은 ATM을 선택해주세요." << endl; }

  for (int i = 0; i < ATM_list.size(); i++) {
    cout << "[" << i << "] ATM " << ATM_list[i]->getSerial() << " [Bank " << ATM_list
[i]->getPrimary()->getBankName() << "/" << ATM_list[i]->getType() << "/" << ATM_list
[i]->getLangType() << "]" << endl;
  }
  cout << endl;
  if (language_setting == 1) { cout << "ATM number : "; }
  if (language_setting == 2) { cout << "ATM 번호 : "; }

  int ATM_choose = 0;
  // cin >> ATM_choose;
  ATM_choose = no_error_range(ATM_list, bank_list, language_setting, 0, (int)ATM_list.
size() - 1);

  //if (ATM_choose >= ATM_list.size()) {
  //  if (language_setting == 1) { cout << "The corresponding ATM does not exist." <<
endl; }
  //  if (language_setting == 2) { cout << "해당하는 ATM이 존재하지 않습니다." << endl; }
  //  cout << "==================== < ATM Service Session End! > ===================="
<< endl;
  //  return;
  //}
  if (language_setting == 1) { cout << ATM_list[ATM_choose]->getSerial() << " ATM is s
elected." << endl << "Connecting to ATM..." << endl; }
  if (language_setting == 2) { cout << ATM_list[ATM_choose]->getSerial() << " ATM이 선택
되었습니다." << endl << "ATM에 접속합니다..." << endl; }

  ATM_list[ATM_choose]->session(bank_list);
  cout << "==================== < ATM Service Session End! > ====================" <<
endl;
  return;
}

Account* BankSearch(vector<ATM*>& ATM_list, vector<Bank*> bank_list, int language_sett
ing) {
  cout << "==================== < Bank Search Session > ====================" << endl;
  if (language_setting == 1) {
    cout << "Please choose searach service number that you want." << endl;
    cout << "[1] By Account Number" << endl;
    cout << "[2] By Card Number" << endl;
    cout << endl;
    cout << "Search service number : ";
  }
  if (language_setting == 2) {
    cout << "원하는 조회 서비스의 번호를 골라주세요.." << endl;
    cout << "[1] 계좌 번호로" << endl;
    cout << "[2] 카드 번호로" << endl;
    cout << endl;
    cout << "Search service number : ";
  }

  int search_choose = 0;
  // cin >> search_choose;
  search_choose = no_error_range(ATM_list, bank_list, language_setting, 1, 2);

  while (true) {
    if (search_choose == 1) {
      string account_number;
      if (language_setting == 1) { cout << "Please write your account number : "; }
      if (language_setting == 2) { cout << "계좌번호를 입력해주세요. : "; }

      //cin >> account_number;
      account_number = Qsearch(ATM_list, bank_list);

      for (int i = 0; i < bank_list.size(); i++) {
        if (bank_list[i]->search_account_number_BankSearch(account_number, language_se
tting) != NULL) {
          cout << "==================== < Bank Search Session End! > =================
===" << endl;
          return bank_list[i]->search_account_number_BankSearch(account_number, langua
ge_setting);
        }
      }
      if (language_setting == 1) { cout << "Account is not found." << endl; }
      if (language_setting == 2) { cout << "계좌를 찾을 수 없습니다." << endl; }
    }
    if (search_choose == 2) {
      string card_number;
      if (language_setting == 1) { cout << "Please write your card number : "; }
      if (language_setting == 2) { cout << "카드 번호를 입력해주세요. : "; }

      //cin >> card_number;
      card_number = Qsearch(ATM_list, bank_list);

      for (int i = 0; i < bank_list.size(); i++) {
        if (bank_list[i]->search_account_card_BankSearch(card_number, language_settin
g) != NULL) {
          cout << "==================== < Bank Search Session End! > =================
===" << endl;
          return bank_list[i]->search_account_card_BankSearch(card_number, language_se
tting);
        }
      }
      if (language_setting == 1) { cout << "Account is not found." << endl; }
      if (language_setting == 2) { cout << "계좌를 찾을 수 없습니다." << endl; }
    }
  }
  cout << "==================== < Bank Search Session End! > ====================" <<
endl;
}

void feeConfig(vector<ATM*>& ATM_list, vector<Bank*> bank_list, int* fee_list1[4], int
* fee_list2[4], int language_setting) {
  if (language_setting == 1) { cout << "Please Enter the deposit fee for primary ban
k." << endl; }
  if (language_setting == 2) { cout << "Primary bank 계좌에서의 입금 수수료를 입력해 주세요." <
< endl; }
  // cin >> *(fee_list1[0]);
  *(fee_list1[0]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the withdrawal fee for primary ba
nk." << endl; }
  if (language_setting == 2) { cout << "Primary bank 계좌에서의 출금 수수료를 입력해 주세요." <
< endl; }
  // cin >> *(fee_list1[1]);
  *(fee_list1[1]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the account transfer fee between
primary banks." << endl; }
  if (language_setting == 2) { cout << "Primary bank 계좌끼리의 송금 수수료를 입력해 주세요." <
< endl; }
  // cin >> *(fee_list1[2]);
  *(fee_list1[2]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the cash transfer fee." << endl;
}
  if (language_setting == 2) { cout << "현금 송금 수수료를 입력해 주세요." << endl; }
  // cin >> *(fee_list1[3]);
  *(fee_list1[3]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the deposit fee for non-primary b
ank." << endl; }
  if (language_setting == 2) { cout << "Non-Primary bank 계좌에서의 입금 수수료를 입력해 주세
요." << endl; }
  // cin >> *(fee_list2[0]);
  *(fee_list2[0]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the withdrawal fee for non-primar
```

```
y bank." << endl; }
  if (language_setting == 2) { cout << "Non-primary bank 계좌에서의 출금 수수료를 입력해 주세
요." << endl; }
  // cin >> *(fee_list2[1]);
  *(fee_list2[1]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the account transfer fee between
primary bank and non-primary banks." << endl; }
  if (language_setting == 2) { cout << "Primary bank 계좌와 non-primary bank 계좌 사이의 송
금 수수료를 입력해 주세요." << endl; }
  // cin >> *(fee_list2[2]);
  *(fee_list2[2]) = no_error(ATM_list, bank_list, language_setting);
  if (language_setting == 1) { cout << "Please Enter the account transfer fee between
non-primary banks." << endl; }
  if (language_setting == 2) { cout << "Non-primary bank 계좌끼리의 송금 수수료를 입력해 주세
요." << endl; }
  // cin >> *(fee_list2[3]);
  *(fee_list2[3]) = no_error(ATM_list, bank_list, language_setting);
  return;
}

int LanguageService(vector<ATM*>& ATM_list, vector<Bank*> bank_list, int language_sett
ing) {
  cout << "===========================<Language Change Session>=======================
====" << endl;
  if (language_setting == 1) {
    cout << "Select language" << endl;
    cout << "[1] English" << endl;
    cout << "[2] 한국어" << endl;
    cout << endl;
    cout << "Please Enter the Number: ";
  }
  if (language_setting == 2) {
    cout << "언어선택" << endl;
    cout << "[1] English" << endl;
    cout << "[2] 한국어" << endl;
    cout << endl;
    cout << "번호 입력 : ";
  }

  int setting = 0;
  // cin >> setting;
  setting = no_error_range(ATM_list, bank_list, language_setting, 1, 2);
  language_setting = setting;

  return language_setting;
}

void Admin(vector<ATM*> ATM_list, vector<Bank*> bank_list, int language_setting) {
  string admin_password = "admin";

  // \t
  if (language_setting == 1) {
    cout << "This is Admin session." << endl;
    cout << endl;
    cout << "Please Enter the password : ";
  }
```

```
    if (language_setting == 2) {
      cout << "관리자 세션입니다." << endl;
      cout << endl;
      cout << "비밀번호를 입력해주세요 : ";
    }
    string input_password;

    int a = 0;
    for (int i = 0; i < 3; i++) {

      //cin >> input_password;
      input_password = Qsearch(ATM_list, bank_list);

      if (admin_password == input_password) {
        if (language_setting == 1) { cout << "Admin confirmed." << endl; }
        if (language_setting == 2) { cout << "관리자 확인 완료." << endl; }
        a = 1;
        break;
      }
      if (a == 0) {
        if (language_setting == 1) { cout << 2 - i << " attempt left." << endl; }
        if (language_setting == 2) { cout << 2 - i << " 회 남음." << endl; }
      }
    }

    if (a == 0) {
      if (language_setting == 1) { cout << "You write wrong password 3 times." << endl;
  }
      if (language_setting == 2) { cout << "잘못된 비밀번호를 3회 입력하셨습니다." << endl; }
      return;
    }

    if (a == 1) {
      if (language_setting == 1) {
        cout << "Please Select the task you want to do." << endl;
        cout << "[1] Bank & Account Info" << endl;
        cout << "[2] ATM Info" << endl;
        cout << endl;
        cout << "Please Enter the Number : ";
      }
      if (language_setting == 2) {
        cout << "진행하고자 하는 업무를 선택하세요." << endl;
        cout << "[1] 은행 & 계좌 정보" << endl;
        cout << "[2] ATM 정보" << endl;
        cout << endl;
        cout << "숫자를 입력해주세요 : ";
      }
    }

    int choice = 0;
    // cin >> choice;
    choice = no_error_range(ATM_list, bank_list, language_setting, 1, 2);

    if (choice == 1) {
      if (language_setting == 1) {
        cout << "==================== < Bank & Account Info > ====================" << e
```

```
ndl;
      for (int i = 0; i < bank_list.size(); i++) {
        cout << endl;
        cout << "Bank : " << bank_list[i]->getBankName() << endl;
        cout << endl;

        for (int j = 0; j < bank_list[i]->get_account().size(); j++) {
          cout << "Name : " << bank_list[i]->get_account()[j]->getUserName() << " ID :
" << bank_list[i]->get_account()[j]->getAccountNumber() << " Password : " << bank_list
[i]->get_account()[j]->getPassword() << " Available Fund : " << bank_list[i]->get_acco
unt()[j]->getAvailableFund() << endl;
          cout << "Card : ";
          for (int k = 0; k < bank_list[i]->get_account()[j]->getCardNumber().size();
k++) {
            cout << bank_list[i]->get_account()[j]->getCardNumber()[k] << "\t";
          }
          cout << endl;
        }
        cout << "=========================================" << endl;
      }
      cout << "==================== < Bank & Account Info End!> ===================="
<< endl;
    }
    if (language_setting == 2) {
      cout << "==================== < Bank & Account Info > ====================" << e
ndl;
      for (int i = 0; i < bank_list.size(); i++) {
        cout << endl;
        cout << "은행 : " << bank_list[i]->getBankName() << endl;
        cout << endl;

        for (int j = 0; j < bank_list[i]->get_account().size(); j++) {
          cout << "예금주 : " << bank_list[i]->get_account()[j]->getUserName() << " 계좌
번호 : " << bank_list[i]->get_account()[j]->getAccountNumber() << " 비밀번호 : " << bank_
list[i]->get_account()[j]->getPassword() << " 계좌잔고 : " << bank_list[i]->get_account
()[j]->getAvailableFund() << endl;
          cout << "카드번호 : ";
          for (int k = 0; k < bank_list[i]->get_account()[j]->getCardNumber().size();
k++) {
            cout << bank_list[i]->get_account()[j]->getCardNumber()[k] << "\t";
          }
          cout << endl;
        }
        cout << "=========================================" << endl;
      }
      cout << "==================== < Bank & Account Info End!> ===================="
<< endl;
    }
  }

  if (choice == 2) {
    if (language_setting == 1) {
      cout << "==================== < ATM Info > ====================" << endl;
      for (int i = 0; i < ATM_list.size(); i++) {
        cout << endl;
        cout << "ATM " << "[" << i << "] " << "Primary Bank : " << ATM_list[i]->getPri
mary()->getBankName() << " " << ATM_list[i]->getType() << " " << ATM_list[i]->getLangT
ype() << endl;
        cout << "Remaing cash : " << 1000 * ATM_list[i]->get1000() + 5000 * ATM_list
[i]->get5000() + 10000 * ATM_list[i]->get10000() + 50000 * ATM_list[i]->get50000() <<
" (1000 : " << ATM_list[i]->get1000() << ", 5000 : " << ATM_list[i]->get5000() << ", 1
0000 : " << ATM_list[i]->get10000() << ", 50000 : " << ATM_list[i]->get50000() << ")"
<< endl;
      }
      cout << "==================== < ATM Info End!> ====================" << endl;
    }
    if (language_setting == 2) {
      cout << "==================== < ATM Info > ====================" << endl;
      for (int i = 0; i < ATM_list.size(); i++) {
        cout << endl;
        cout << "ATM " << "[" << i << "] " << "Primary Bank : " << ATM_list[i]->getPri
mary()->getBankName() << " " << ATM_list[i]->getType() << " " << ATM_list[i]->getLangT
ype() << endl;
        cout << "잔여현금 : " << 1000 * ATM_list[i]->get1000() + 5000 * ATM_list[i]->get
5000() + 10000 * ATM_list[i]->get10000() + 50000 * ATM_list[i]->get50000() << " (1000
: " << ATM_list[i]->get1000() << ", 5000 : " << ATM_list[i]->get5000() << ", 10000 : "
<< ATM_list[i]->get10000() << ", 50000 : " << ATM_list[i]->get50000() << ")" << endl;
      }
      cout << "==================== < ATM Info End!> ====================" << endl;
    }
  }
}

//-------------------------------------------------------------------main Function Ca
ll
int main() {
  vector<Bank*> bank_list;
  vector<ATM*> ATM_list;
  vector<Bank*>* blist = &bank_list;
  vector<ATM*>* alist = &ATM_list;

  int* fee1[4] = { 0, };
  int* fee2[4] = { 0, };
  for (int i = 0; i < 4; i++) {
    fee1[i] = new int(0);
  }
  for (int i = 0; i < 4; i++) {
    fee2[i] = new int(0);
  }

  bool onSession = true;
  int language_setting = 1; // 1: English, 2: Korean

  while (onSession) {
    cout << "==========================<Welcome to Bank System Service>==============
=============" << endl;
    if (language_setting == 1) {
      cout << "Please Select the task you want to do." << endl;
      cout << "[1] Make Bank" << endl;
      cout << "[2] Banking Service (Make Account or Make Card)" << endl;
      cout << "[3] Make ATM" << endl;
      cout << "[4] ATM Service (Deposit, Withdraw, etc...)" << endl;
      cout << "[5] Change Language Setting" << endl;
      cout << "[6] Shut Down the Bank System Service" << endl;
      cout << "[7] Fee Configuration" << endl;
      cout << "[8] Admin" << endl;
      cout << endl;

      cout << "Please Enter the Number : ";
    }
    if (language_setting == 2) {
      cout << "진행하고자 하는 업무를 선택하세요." << endl;
      cout << "[1] 은행 만들기" << endl;
      cout << "[2] 은행 서비스 (계좌 및 카드 생성)" << endl;
      cout << "[3] ATM 만들기" << endl;
      cout << "[4] ATM 서비스 (입금, 출금 등...)" << endl;
      cout << "[5] 언어 설정 변경" << endl;
      cout << "[6] Bank System Service 종료" << endl;
      cout << "[7] 수수료 설정" << endl;
      cout << "[8] 관리자" << endl;
      cout << endl;
      cout << "숫자를 입력해주세요 : ";
    }

    int session_chice = 0;
    session_chice = no_error_range(ATM_list, bank_list, language_setting, 1, 8);

    switch (session_chice) {
    case 1:
      BankMake(ATM_list, bank_list, language_setting, blist, alist);
      break;
    case 2:
      BankService(ATM_list, bank_list, language_setting);
      break;
    case 3:
      ATMMake(ATM_list, bank_list, fee1, fee2, language_setting, blist, alist);
      break;
    case 4:
      ATMService(ATM_list, bank_list, language_setting);
      break;
    case 5:
      language_setting = LanguageService(ATM_list, bank_list, language_setting);
      break;
    case 6:
      onSession = false;
      break;
    case 7:
      feeConfig(ATM_list, bank_list, fee1, fee2, language_setting);
      break;
    case 8:
      Admin(ATM_list, bank_list, language_setting);
    }
    cout << "===========================<End System Session>==========================
=" << endl;
  }

  cout << "===========================<End System>===========================" << end
l;
```

```
  for (int i = 0; i < 4; i++) {
    delete fee1[i];
  }
  for (int i = 0; i < 4; i++) {
    delete fee2[i];
  }
  for (int i = 0; i < (int) bank_list.size(); i++) {
    delete bank_list[i];
  }
  for (int i = 0; i < (int)ATM_list.size(); i++) {
    delete ATM_list[i];
  }

  return 0;
}
```

6. Member Student Contribution

Student ID	Student Name	Contribution Percentage
201811043	김태형	All students equally contributed
201811147	이지환	All students equally contributed
201811195	한도희	All students equally contributed