**- Automated Teller Machine (ATM) System –**

|  |  |
| --- | --- |
| 201511047 | Kim, Jong-Soo |
| 201611030 | Kim, San-Ha |
| 201911012 | Kong, Young-Jae |
| 201911088 | An, Sang-Jun |

목차

[1. The final class Diagram design - 1 -](#_Toc89803668)

[1.1 Diagram design - 1 -](#_Toc89803669)

[2. The list of requirements - 1 -](#_Toc89803670)

[2.1 System Setup - 1 -](#_Toc89803671)

[2.2 ATM Session - 1 -](#_Toc89803672)

[2.3 User Authorization - 1 -](#_Toc89803673)

[2.4 Deposit - 2 -](#_Toc89803674)

[2.5 Withdrawal - 2 -](#_Toc89803675)

[2.6 Transfer - 2 -](#_Toc89803676)

[2.7 Display of Transaction History (Admin Menu) - 3 -](#_Toc89803677)

[2.8 Multi-language support - 3 -](#_Toc89803678)

[2.9 Exception Handling - 3 -](#_Toc89803679)

[3. The console screenshots - 4 -](#_Toc89803680)

[3.1 System Setup - 4 -](#_Toc89803681)

[3.2 ATM Session - 5 -](#_Toc89803682)

[3.3 User Authorization - 5 -](#_Toc89803683)

[3.4 Deposit - 6 -](#_Toc89803684)

[3.5 Withdrawal - 6 -](#_Toc89803685)

[3.6 Transfer - 6 -](#_Toc89803686)

[3.7 Display of Transaction History (Admin Menu) - 7 -](#_Toc89803687)

[3.8 Multi-language support - 7 -](#_Toc89803688)

[3.9 Exception Handling - 7 -](#_Toc89803689)

[1. The list of concepts of object-oriented programming - 7 -](#_Toc89803690)

[2. Instruction to run the source code - 7 -](#_Toc89803691)

[3. The final version of source code - 7 -](#_Toc89803692)

[7. Member student contribution table and note - 40 -](#_Toc89803693)

# 1. The final class Diagram design

## 1.1 Diagram design

# 2. The list of requirements

## 2.1 System Setup

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

## 2.2 ATM Session

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

## 2.3 User Authorization

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

## 2.4 Deposit

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

## 2.5 Withdrawal

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

## 2.6 Transfer

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

## 2.7 Display of Transaction History (Admin Menu)

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

## 2.8 Multi-language support

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

## 2.9 Exception Handling

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

# 3. The console screenshots

## 3.1 System Setup

* (REQ1.1) An ATM has a serial number that can be uniquely identified among all ATMs.   
  텍스트이(가) 표시된 사진

  자동 생성된 설명  
  테이블이(가) 표시된 사진

  자동 생성된 설명
* (REQ1.2) An ATM is set to one of the following types: (1) Single Bank ATM, (2) Multi-Bank ATM.   
    
  텍스트이(가) 표시된 사진

  자동 생성된 설명텍스트이(가) 표시된 사진

  자동 생성된 설명  
  테이블이(가) 표시된 사진

  자동 생성된 설명테이블이(가) 표시된 사진

  자동 생성된 설명
* (REQ1.3) An ATM may support either unilingual or bilingual languages.
* (REQ1.4) A Bank deposits a certain amount of cashes to an ATM to serve users.
* (REQ1.5) A Bank can open an Account for user with necessary information to perform bank services.
* (REQ1.6) A user may have multiple Accounts in a Bank.
* (REQ1.7) A user may have Accounts in multiple Banks.
* (REQ1.8) Each ATM have several types of transaction fees as follows
* (REQ1.9) An admin can access the menu of “Transaction History” via an admin card (See REQ Display of Transaction History).

## 3.2 ATM Session

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

## 3.3 User Authorization

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

## 3.4 Deposit

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

## 3.5 Withdrawal

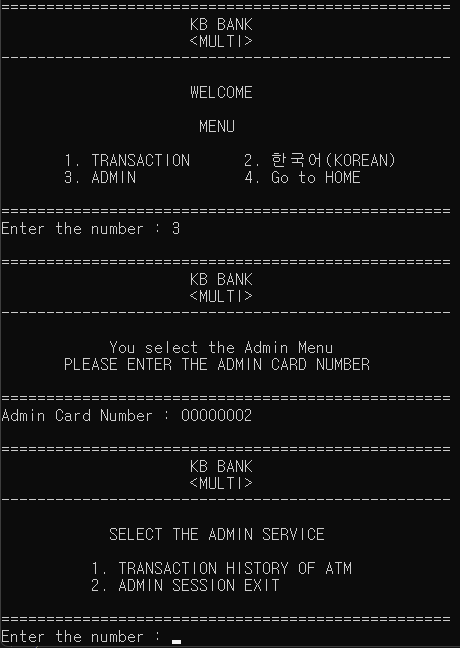
* (REQ5.1) An ATM shall ask a user to enter the amount of fund to withdraw.
* (REQ5.2) An ATM shall display an appropriate error message if there is insufficient fund in the account or insufficient cash in the ATM.
* (REQ5.3) Once the withdrawal is successful, the transaction must be reflected to the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

## 3.6 Transfer

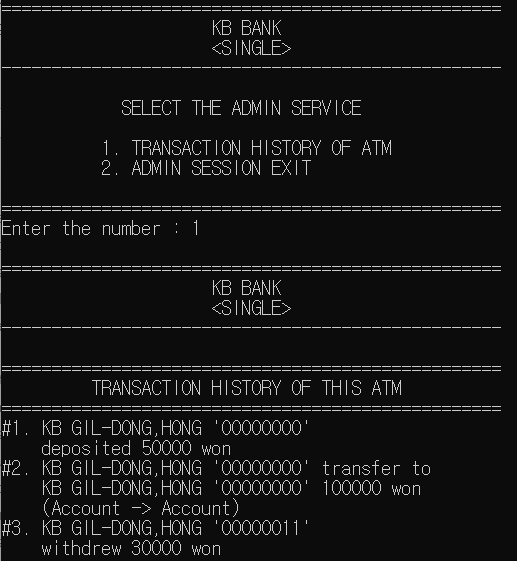
* (REQ6.1) An ATM shall ask a user to choose the types of transfer either cash transfer or account fund transfer.
* (REQ6.2) For both cash and account transfers, an ATM shall ask the destination account number where the fund is to be transferred.
* (REQ6.3) For cash transfer, an ATM shall ask user to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

## 3.7 Display of Transaction History (Admin Menu)

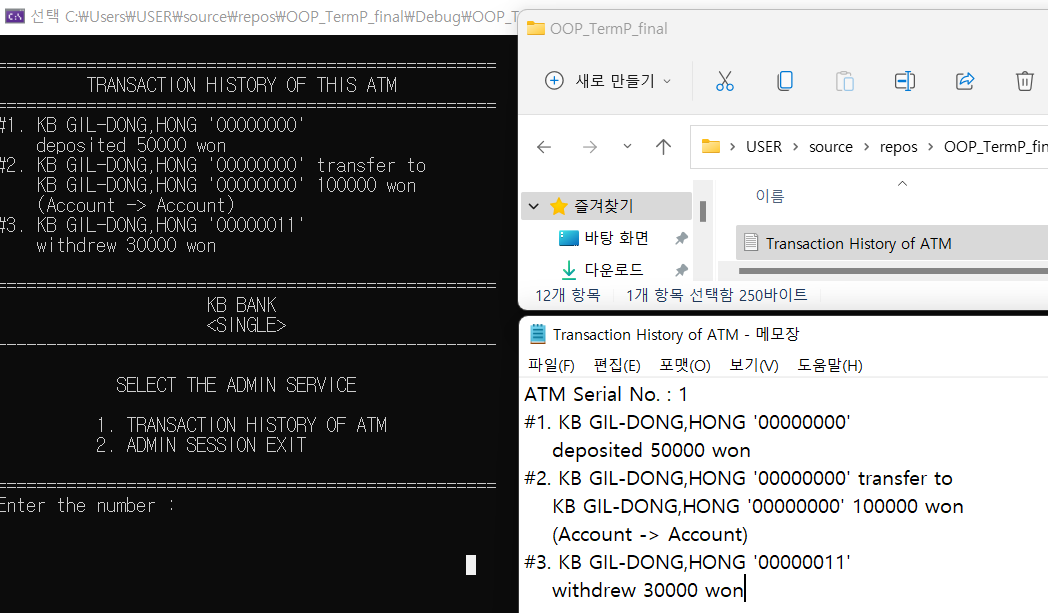
* (REQ7.1) When a session is started by an admin by inserting an admin card (See REQ in System Setup), an ATM displays a menu of “Transaction History” only.



* (REQ7.2) When the “Transaction History” menu is selected, an ATM display the information of all transactions from all users from the beginning of the system start



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



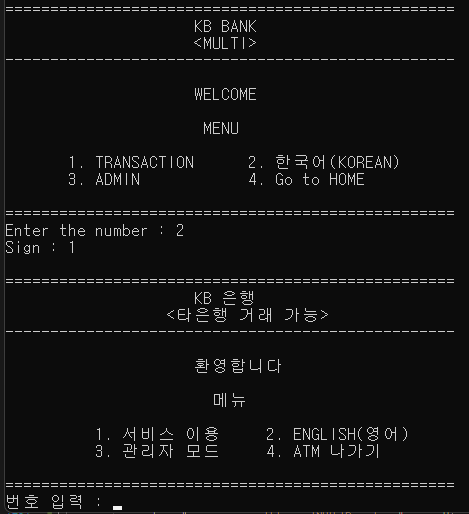
## 3.8 Multi-language support

* (REQ8.1) An ATM that is configured with the bilingual support shall provide an option for a user to choose the preferred language either English or Korean.

#default = English



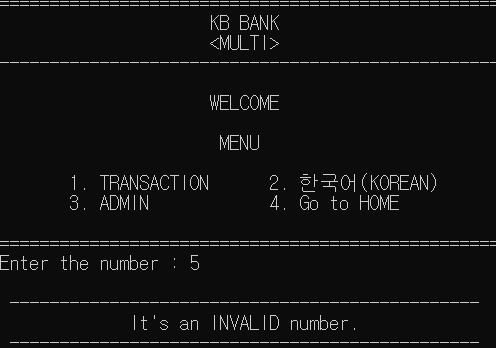
* (REQ8.2) Once a certain language is chosen, all menus must be displayed using the chosen language.



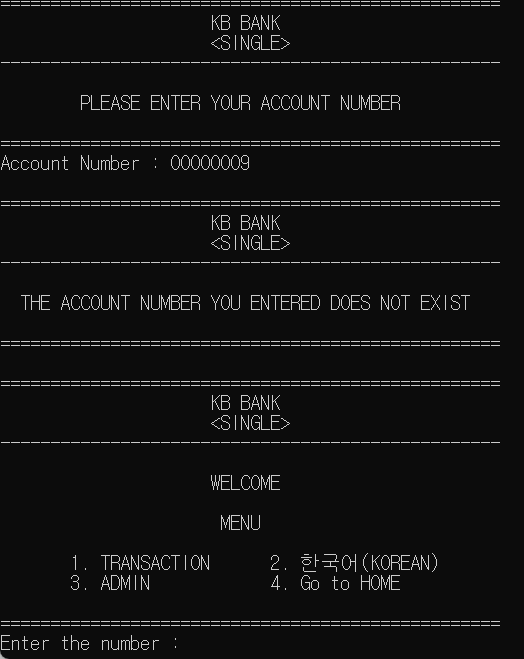
## 3.9 Exception Handling

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

#In ATM Session, If you write a wrong number, It returns “Invalid number”



#If you enter an account number that does not exist, you will return to the previous session with the phrase "ACCOUNT NUMBER YOU ENTERED DOES NOT EXIST".



# The list of concepts of object-oriented programming

# Instruction to run the source code

# The final version of source code

|  |
| --- |
| 1. */\** 2. *\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 3. *\*\*\* \* \** 4. *\*\*\* \* Object-Oriented Programming Course Term Project \** 5. *\*\*\* \* \** 6. *\*\*\* \* 201511047 Kim,Jong-Soo 201611030 Kim,San-Ha \** 7. *\*\*\* \* 201911012 Kong,Young-Jae 201911088 An,Sang-Jun \** 8. *\*\*\* \* \** 9. *\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 10. *\*/* 11. #include *<iostream>* 12. #include *<fstream>* 13. #include *<vector>* 14. #include *<string>* 15. #include *<sstream>* 16. #include *<iomanip>* 17. **using** **namespace** **std**; 18. */\** 19. *\*\*\* Class Building* 20. *\*/* 21. */\*--------------------------------------- Account Class ---------------------------------------\*/* 22. **class** **Transaction**; *// 전방선언* 23. **class** **Account** { 24. **protected**: 25. string accountNumber = ""; 26. string accountPw = ""; 27. string userName = ""; 28. unsigned long long availableFunds = 0; 29. vector<Transaction> transactionHistoryOfAccount; 30. **public**: 31. Account() {} 32. Account(string aNum, string aPW, string uName, string funds) { 33. accountNumber = aNum; 34. accountPw = aPW; 35. userName = uName; 36. availableFunds = stoull(funds); 37. } 38. ~Account() {} 39. string getAccountNumber() { **return** accountNumber; } 40. bool check\_pw(string inPw) { 41. **if** (accountPw.compare(inPw) == 0) **return** true; 42. **else** **return** false; 43. } 44. void plusMoney(double amount) { availableFunds += amount; } 45. void minusMoney(double amount) { availableFunds -= amount; } 46. bool compareAccount(string account) { 47. **if** (accountNumber.compare(account) == 0) **return** true; 48. **else** **return** false; 49. } 50. void addTransaction(Transaction\* trans) {transactionHistoryOfAccount.push\_back(\*trans);} 51. unsigned long long getFundInfo() {**return** availableFunds;} 52. string getInfo() {**return** userName;} 53. vector<Transaction> getTransactionHistoryOfAccount() {**return** transactionHistoryOfAccount;} 54. }; 55. */\*------------------------------------------------ Transaction Class ---------------------------------------------------\*/* 56. */\*--------------거래내역 남기는 법: Session에서 거래 시작 시 Transaction\* transaction 선언 후 거래 종료 시,--------------\*/* 57. */\*---transaction = new {deposit, withdraw, transfer 중 1}Transaction(transaction, (dest account), account, amount)---\*/* 58. *//class Transaction : public Account {* 59. **class** **Transaction** { 60. **protected**: 61. **static** unsigned long long transactionCnt; 62. unsigned long long ID = transactionCnt; 63. unsigned long long Amount = 0; 64. string englishDescription = "#"; 65. string koreanDescription = "#"; 66. Account\* account; 67. **public**: 68. unsigned long long getID() { **return** ID; } 69. unsigned long long getAmount() { **return** Amount; } 70. string getEnglishInformation() {**return** englishDescription;} 71. string getKoreanInformation() {**return** koreanDescription;} 72. }; 73. unsigned long long Transaction::transactionCnt{ 1 }; 74. */\*------------ Children Classes of Transaction Class ------------\*/* 75. */\*-------- Deposit Transaction Class --------\*/* 76. **class** **DepositTransaction** : **public** Transaction { 77. **public**: 78. DepositTransaction(Account\* acc, unsigned long long amount, string accountBank); 79. *// string getInformation();* 80. }; 81. DepositTransaction::DepositTransaction(Account\* acc, unsigned long long amount, string accountBank) { 82. account = acc; 83. Amount = amount; 84. ID = transactionCnt++; 85. englishDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'**\n** deposited " + to\_string(amount) + " won"); 86. koreanDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'에**\n** " + to\_string(amount) + " 원이 입금되었습니다"); 87. } 88. *//string DepositTransaction::getInformation() {* 89. *// return description;* 90. *//} /\* ex. 1 1234 5678 deposited 100$ \*/* 91. *// /\* ex. 2 1234 5678 withdrew 50$ \*/* 92. */\*------- Withdrawal Transaction Class ------\*/* 93. **class** **WithdrawalTransaction** : **public** Transaction { 94. **public**: 95. WithdrawalTransaction(Account\* acc, unsigned long long amount, string accountBank); 96. *// string getInformation();* 97. }; 98. WithdrawalTransaction::WithdrawalTransaction(Account\* acc, unsigned long long amount, string accountBank) { 99. account = acc; 100. Amount = amount; 101. ID = transactionCnt++; 102. englishDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'**\n** withdrew " + to\_string(amount) + " won"); 103. koreanDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'에서**\n** " + to\_string(amount) + " 원이 출금되었습니다"); 104. } 105. *//string WithdrawalTransaction::getInformation() {* 106. *// return description;* 107. *//}* 108. */\*-------- Transfer Transaction Class -------\*/* 109. **class** **TransferTransaction** : **public** Transaction { 110. **protected**: 111. Account\* destaccount; 112. }; 113. **class** **AccountTransferTransaction** : **public** TransferTransaction { 114. **public**: 115. AccountTransferTransaction(Account\* destacc, Account\* account, unsigned long long amount, string accountBank, string destBank); 116. *// string getInformation();* 117. }; 118. AccountTransferTransaction::AccountTransferTransaction(Account\* destacc, Account\* acc, unsigned long long amount, string accountBank, string destBank) { 119. destaccount = destacc; 120. account = acc; 121. Amount = amount; 122. ID = transactionCnt++; 123. englishDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "' transfer to**\n** " + destBank + " " + destaccount->getInfo() + " '" + account->getAccountNumber() + "' " + to\_string(amount) + " won**\n** (Account -> Account)"); 124. koreanDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'에서**\n** " + destBank + " " + destaccount->getInfo() + " '" + account->getAccountNumber() + "'로**\n** " + to\_string(amount) + " 원을 계좌 송금하였습니다"); 125. } 126. *//string AccountTransferTransaction::getInformation() {* 127. *// return description;* 128. *//}* 129. **class** **CashTransferTransaction** : **public** TransferTransaction { 130. **public**: 131. CashTransferTransaction(Account\* destaccount, Account\* account, unsigned long long amount, string accountBank, string destBank); 132. *// string getInformation();* 133. }; 134. CashTransferTransaction::CashTransferTransaction(Account\* destacc, Account\* acc, unsigned long long amount, string accountBank, string destBank) { 135. destaccount = destacc; 136. Amount = amount; 137. ID = transactionCnt++; 138. englishDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "' transfer to**\n** " + destBank + " " + destaccount->getInfo() + " '" + account->getAccountNumber() + "' " + to\_string(amount) + " won**\n** (Cash -> Account))"); 139. koreanDescription.append(to\_string(ID) + ". " + accountBank + " " + account->getInfo() + " '" + account->getAccountNumber() + "'에서**\n** " + destBank + " " + destaccount->getInfo() + " '" + account->getAccountNumber() + "'로**\n** " + to\_string(amount) + " 원을 현금 송금하였습니다"); 140. } 141. *//string CashTransferTransaction::getInformation() {* 142. *// return description;* 143. *//}* 144. */\*----------------------------------------- Bank Class ----------------------------------------\*/* 145. **class** **Bank** { 146. **private**: 147. string bankName; 148. vector<Account> accounts; 149. **public**: 150. Bank() { 151. bankName = "unknown"; 152. } 153. Bank(string name) { 154. bankName = name; 155. } 156. string getBankName() {**return** bankName;} 157. Account\* findAccountOfBank(string account); 158. void addAccount(Account account) {accounts.push\_back(account);} 159. }; 160. Account\* Bank::findAccountOfBank(string account) { 161. int i; 162. **for** (i = 0; i < accounts.size(); i++) { 163. **if** (accounts[i].compareAccount(account)) { 164. **return** &accounts[i]; 165. **break**; 166. } 167. } 168. **return** **nullptr**; 169. } 170. vector<Bank> bankData; 171. Bank\* findBank(string name) { 172. int i; 173. **for** (i = 0; i < bankData.size(); i++) { 174. **if** (bankData[i].getBankName().compare(name) == 0) { 175. **return** &bankData[i]; 176. **break**; 177. } 178. } 179. **return** **nullptr**; 180. } 181. Bank\* findAccount(string account) { 182. **for** (int i = 0; i < bankData.size(); i++) { 183. **if** (bankData[i].findAccountOfBank(account) != **nullptr**) { 184. **return** &bankData[i]; 185. **break**; 186. } 187. } 188. **return** **nullptr**; 189. } 190. */\*--------------------------------------- Session Class ---------------------------------------\*/* 191. **class** **ATM**; 192. **class** **Session** { 193. **protected**: 194. ATM\* atm; 195. Account\* account; 196. vector<Transaction> transactionHistoryOfSession; 197. bool authorizationSignal; 198. int authorizationCount; 199. int withdrawalCount; 200. bool primarySignal; *// 현재 account의 은행 정보와 ATM의 주거래 은행이 동일하면 true* 201. **public**: 202. Session() {} 203. void CashDeposit(unsigned long long amount, int x); 204. void CheckDeposit(unsigned long long amount, int x); 205. void Withdrawal(unsigned long long amount, int x); 206. void CashTransfer(unsigned long long amount, Account\* destination, int x); 207. void AccountTransfer(unsigned long long amount, Account\* destination, int x); 208. bool Authorization(string password) {**return** account->check\_pw(password);} 209. }; 210. */\*----------------------------------------- ATM Class -----------------------------------------\*/* 211. **class** **ATM** { 212. **protected**: 213. **static** int atmCnt; 214. int serialNum; *// ATM 시리얼 넘버* 215. Bank\* primaryBank; *// 주거래 은행* 216. unsigned long long cashAmount; *// ATM에 들어있는 현금* 217. vector<vector<Transaction>> transactionHistoryOfATM; *// ATM 전체 transaction history (Admin에서 접근 가능)* 218. string AdminNum; *// Admin 넘버* 219. Session session; *// 세션* 220. int SingleOrMulti; *// 0: Single , 1: Multi* 221. **public**: 222. ATM() {} 223. string getPrimaryBankInfo() {**return** primaryBank->getBankName();} 224. int getSerialNum() {**return** serialNum;} 225. Bank\* getPrimaryBank() {**return** primaryBank;} 226. unsigned long long getCashAmount() {**return** cashAmount;} 227. int getSingleInfo() {**return** SingleOrMulti;} 228. void addTransaction(vector<Transaction> transac) {transactionHistoryOfATM.push\_back(transac);} 229. void plusMoney(unsigned long long amount) {cashAmount += amount;} 230. void minusMoney(unsigned long long amount) {cashAmount -= amount;} 231. void startKoreanAdminSession(); 232. void startEnglishAdminSession(); 233. void mainEnglishDisplay() { 234. cout << "**\n**==================================================" << endl; 235. cout << " " << **this**->getPrimaryBankInfo() << " BANK" << endl; 236. cout << " <"; 237. **if** (**this**->getSingleInfo() == 0) cout << "SINGLE>" << endl; 238. **else** cout << "MULTI>" << endl; 239. cout << "--------------------------------------------------**\n**" << endl; 240. } 241. void mainKoreanDisplay() { 242. cout << "**\n**==================================================" << endl; 243. cout << " " << **this**->getPrimaryBankInfo() << " 은행" << endl; 244. cout << " <"; 245. **if** (**this**->getSingleInfo() == 0) cout << "주거래 은행 전용>" << endl; 246. **else** cout << "타은행 거래 가능>" << endl; 247. cout << "--------------------------------------------------**\n**" << endl; 248. } 249. void invalidEnglishDisplay() { 250. cout << "**\n** ----------------------------------------------- " << endl; 251. cout << " It's an INVALID number." << endl; 252. cout << " ----------------------------------------------- **\n**" << endl; 253. cout << "==================================================" << endl; 254. } 255. void invalidKoreanDisplay() { 256. cout << "**\n** ----------------------------------------------- " << endl; 257. cout << " 유효하지 않은 번호입니다." << endl; 258. cout << " ----------------------------------------------- **\n**" << endl; 259. cout << "==================================================" << endl; 260. } 261. **virtual** void startSession() = 0; 262. **virtual** string getClassName() = 0; 263. **virtual** ~ATM() = **default**; 264. }; 265. int ATM::atmCnt{ 1 }; 266. vector<ATM\*> atmData; 267. */\*-------------- Methods of Session Class --------------\*/* 268. void Session::CashDeposit(unsigned long long amount, int x) { 269. unsigned long long fee = 0; 270. **if** (!primarySignal) fee = 500; 271. atm->plusMoney(amount); 272. account->plusMoney(amount - fee); 273. DepositTransaction newTransaction(account, amount, findAccount(account->getAccountNumber())->getBankName()); 274. account->addTransaction(&newTransaction); 275. transactionHistoryOfSession.push\_back(newTransaction); 276. **if** (x == 0) cout << newTransaction.getKoreanInformation() << endl; 277. **else** cout << newTransaction.getEnglishInformation() << endl; 278. **if** (x == 0) cout << "**\n**현재 잔액 : "; 279. **else** cout << "**\n**Current Available Funds : "; 280. cout << account->getFundInfo(); 281. **if** (x == 0) cout << " 원" << endl; 282. **else** cout << " won" << endl; 283. cout << "**\n**"; 284. } 285. void Session::CheckDeposit(unsigned long long amount, int x) { 286. unsigned long long fee = 0; 287. **if** (!primarySignal) fee = 500; 288. account->plusMoney(amount - fee); 289. DepositTransaction newTransaction(account, amount, findAccount(account->getAccountNumber())->getBankName()); 290. account->addTransaction(&newTransaction); 291. transactionHistoryOfSession.push\_back(newTransaction); 292. **if** (x == 0) cout << newTransaction.getKoreanInformation() << endl; 293. **else** cout << newTransaction.getEnglishInformation() << endl; 294. **if** (x == 0) cout << "**\n**현재 잔액 : "; 295. **else** cout << "**\n**CURRENT BALANCE : "; 296. cout << account->getFundInfo(); 297. **if** (x == 0) cout << " 원" << endl; 298. **else** cout << " won" << endl; 299. cout << "**\n**"; 300. } 301. void Session::Withdrawal(unsigned long long amount, int x) { 302. unsigned long long fee = 500; 303. **if** (!primarySignal) fee = 1000; 304. **if** (atm->getCashAmount() < amount) { 305. **if** (x == 0) cout << " 현재 기기 내 현금이 부족합니다**\n**" << endl; 306. **else** cout << " OUR ATM DOESN'T HAVE ENOUGH MONEY**\n**" << endl; 308. } 309. **else** **if** (amount + fee > account->getFundInfo()) { 310. **if** (x == 0) cout << " 잔액 부족**\n**" << endl; 311. **else** cout << " YOU DON'T HAVE ENOUGH MONEY**\n**" << endl; 312. } 313. **else** { 314. atm->minusMoney(amount); 315. account->minusMoney(amount + fee); 316. WithdrawalTransaction newTransaction(account, amount, findAccount(account->getAccountNumber())->getBankName()); 317. account->addTransaction(&newTransaction); 318. transactionHistoryOfSession.push\_back(newTransaction); 319. **if** (x == 0) cout << newTransaction.getKoreanInformation() << endl; 320. **else** cout << newTransaction.getEnglishInformation() << endl; 321. withdrawalCount ++; 322. **if** (x == 0) cout << "**\n**현재 잔액 : "; 323. **else** cout << "**\n**CURRENT BALANCE : "; 324. cout << account->getFundInfo(); 325. **if** (x == 0) cout << " 원" << endl; 326. **else** cout << " won" << endl; 327. cout << "**\n**"; 328. } 329. } 330. void Session::CashTransfer(unsigned long long amount, Account\* destination, int x) { 331. atm->plusMoney(amount); 332. destination->plusMoney(amount); 333. CashTransferTransaction newTransaction(destination, account, amount, findAccount(account->getAccountNumber())->getBankName(), findAccount(destination->getAccountNumber())->getBankName()); 334. destination->addTransaction(&newTransaction); 335. transactionHistoryOfSession.push\_back(newTransaction); 336. **if** (x == 0) cout << newTransaction.getKoreanInformation() << endl; 337. **else** cout << newTransaction.getEnglishInformation() << endl; 338. **if** (x == 0) cout << "**\n**현재 잔액 : "; 339. **else** cout << "**\n**CURRENT BALANCE : "; 340. cout << account->getFundInfo(); 341. **if** (x == 0) cout << " 원" << endl; 342. **else** cout << " won" << endl; 343. cout << "**\n**"; 344. } 345. void Session::AccountTransfer(unsigned long long amount, Account\* destination, int x) { 346. unsigned long long fee; 347. string accountNum = (findAccount(account->getAccountNumber()))->getBankName(); 348. string destNum = (findAccount(destination->getAccountNumber()))->getBankName(); 349. **if** ( (accountNum.compare(destNum) == 0) && (primarySignal == true) ) fee = 1500; 350. **else** **if** ( (accountNum.compare(destNum) == 0) && (primarySignal == false) ) fee = 2500; 351. **else** fee = 2000; 352. **if** (amount + fee > account->getFundInfo()) { 353. **if** (x == 0) cout << " 잔액 부족**\n**" << endl; 354. **else** cout << " YOU DON'T HAVE ENOUGH MONEY**\n**" << endl; 355. } **else** { 356. account->minusMoney(amount + fee); 357. destination->plusMoney(amount); 358. AccountTransferTransaction newTransaction(destination, account, amount, accountNum, destNum); 359. destination->addTransaction(&newTransaction); 360. account->addTransaction(&newTransaction); 361. transactionHistoryOfSession.push\_back(newTransaction); 362. **if** (x == 0) cout << newTransaction.getKoreanInformation() << endl; 363. **else** cout << newTransaction.getEnglishInformation() << endl; 364. **if** (x == 0) cout << "**\n**현재 잔액 : "; 365. **else** cout << "**\n**CURRENT BALANCE : "; 366. cout << account->getFundInfo(); 367. **if** (x == 0) cout << " 원" << endl; 368. **else** cout << " won" << endl; 369. cout << "**\n**"; 370. } 371. } 372. */\*-------------- Children Classes of Session Class --------------\*/* 373. */\*---------- Korean Session Class -----------\*/* 374. **class** **KoreanSession** : **public** Session { 375. **public**: 376. KoreanSession(ATM\* iatm) { 377. atm = iatm; 378. primarySignal = true; 379. authorizationCount = 0; 380. withdrawalCount = 0; 381. authorizationSignal = true; 382. bool validAccount = true; 383. string inputAccount; 384. atm->mainKoreanDisplay(); 385. cout << " 계좌 번호를 입력해주세요**\n**" << endl; 386. cout << "==================================================" << endl; 387. cout << "계좌 번호 : "; 388. cin >> inputAccount; 390. **if** (findAccount(inputAccount) == **nullptr**) { 391. atm->mainKoreanDisplay(); 392. cout << " 입력한 계좌번호가 존재하지 않습니다**\n**" << endl; 393. cout << "==================================================" << endl; 394. validAccount = false; 395. } 396. **else** { 397. Bank\* temp = findAccount(inputAccount); 398. **if** ( (atm->getPrimaryBankInfo()).compare(temp->getBankName()) == 0 ) { 399. account = temp->findAccountOfBank(inputAccount); 400. } **else** { 401. **if** (atm->getSingleInfo() == 0) { 402. atm->mainKoreanDisplay(); 403. cout << " 타은행 계좌는 사용하실 수 없습니다**\n**" << endl; 404. cout << "==================================================" << endl; 405. validAccount = false; 406. } **else** { 407. primarySignal = false; 408. account = temp->findAccountOfBank(inputAccount); 409. } 410. } 411. } 412. **if** (validAccount) { 413. **for** (int i = 1; i < 4; i++) { 414. string inputPassword; 415. atm->mainKoreanDisplay(); 416. cout << " 비밀번호를 입력해주세요**\n**" << endl; 417. cout << "==================================================" << endl; 418. cout << "비밀번호 : "; 419. cin >> inputPassword; 420. **if** (Authorization(inputPassword)) { 421. authorizationSignal = true; 422. **break**; 423. } 424. **else** { 425. authorizationSignal = false; 426. authorizationCount ++; 427. atm->mainKoreanDisplay(); 428. cout << " 비밀번호를 " << authorizationCount << "회 틀렸습니다**\n**" << endl; 429. cout << "==================================================" << endl; 430. } 431. } 432. **if** (authorizationSignal == false) { 433. atm->mainKoreanDisplay(); 434. cout << " 비밀번호 입력을 3회 실패하여 세션이 종료됩니다**\n**" << endl; 435. cout << "==================================================" << endl; 436. } 437. **else** { 438. bool sessionExitSignal = true; 439. **while** (sessionExitSignal) { 440. atm->mainKoreanDisplay(); 441. cout << " 원하시는 서비스를 선택해주세요**\n**" << endl; 442. cout << " 1. 입금 2. 출금" << endl; 443. cout << " 3. 송금 4. 거래 내역 조회" << endl; 444. cout << " 5. 서비스 종료**\n**" << endl; 445. cout << "==================================================" << endl; 446. cout << "번호 입력 : "; 447. int transactionNum = -1; 448. cin >> transactionNum; 449. **if** (cin.fail() == true) { 450. atm->invalidKoreanDisplay(); 451. cin.clear(); 452. cin.ignore(100, '\n'); 453. **continue**; 454. }  457. **if** (transactionNum == 1) { *// Deposit* 458. atm->mainKoreanDisplay(); 459. cout << " 입금 서비스를 선택하셨습니다**\n**" << endl; 460. cout << " 현금과 수표 중 하나를 선택하십시오**\n**" << endl; 461. cout << " 1. 현금 입금 2. 수표 입금**\n**" << endl; 462. cout << "==================================================" << endl; 463. cout << "번호 입력 : "; 464. int x = -1; 465. cin >> x; 466. **if** (cin.fail() == true) { 467. atm->invalidKoreanDisplay(); 468. cin.clear(); 469. cin.ignore(100, '\n'); 470. **continue**; 471. } 472. unsigned long long inAmount; 473. **if** (x == 1) { 474. **while** (true) { 475. atm->mainKoreanDisplay(); 476. cout << " 입금하실 1만원권 지폐의 장 수를 입력해주세요**\n**" << endl; 477. cout << "==================================================" << endl; 478. cout << "1만원권 지폐 장 수 : "; 479. int numBill = -1; 480. cin >> numBill; 481. **if** (cin.fail() == true) { 482. atm->invalidKoreanDisplay(); 483. cin.clear(); 484. cin.ignore(100, '\n'); 485. **continue**; 486. } 487. **if** ((0 < numBill) && (numBill <= 50)) {inAmount = 10000 \* numBill; **break**;} 488. **else** **if** (numBill > 50) { 489. atm->mainKoreanDisplay(); 490. cout << " 거래 1회 당 입금 가능한 장 수를 초과하셨습니다**\n**" << endl; 491. cout << "==================================================" << endl; 492. } **else** atm->invalidKoreanDisplay(); 493. } 494. atm->mainKoreanDisplay(); 495. CashDeposit(inAmount, 0); 496. cout << "==================================================" << endl; 497. } **else** **if** (x == 2) { 498. **while** (true) { 499. atm->mainKoreanDisplay(); 500. cout << " 입금하실 10만원권 수표의 장 수를 입력해주세요**\n**" << endl; 501. cout << "==================================================" << endl; 502. cout << "10만원권 수표 장 수 : "; 503. int numBill = -1; 504. cin >> numBill; 505. **if** (cin.fail() == true) { 506. atm->invalidKoreanDisplay(); 507. cin.clear(); 508. cin.ignore(100, '\n'); 509. **continue**; 510. } 511. **if** ((0 < numBill) && (numBill <= 30)) {inAmount = 100000 \* numBill; **break**;} 512. **else** **if** (numBill > 30) { 513. atm->mainKoreanDisplay(); 514. cout << " 거래 1회 당 입금 가능한 장 수를 초과하셨습니다**\n**" << endl; 515. cout << "==================================================" << endl; 516. } **else** atm->invalidKoreanDisplay(); 517. } 518. atm->mainKoreanDisplay(); 519. CheckDeposit(inAmount, 0); 520. cout << "==================================================" << endl; 521. } **else** atm->invalidKoreanDisplay();  524. } **else** **if** (transactionNum == 2) { *// Withdrawal* 525. **if** (withdrawalCount == 3) { 526. atm->mainKoreanDisplay(); 527. cout << " 세션 1회 당 출금 가능한 액수를 초과하셨습니다**\n**" << endl; 528. cout << "==================================================" << endl; 529. } 530. **else** { 531. long long inAmount = -1; 532. **while** (true) { 533. atm->mainKoreanDisplay(); 534. cout << " 출금 서비스를 선택하셨습니다**\n**" << endl; 535. cout << " 출금하실 금액을 선택해주십시오**\n**" << endl; 536. cout << "1. 10,000 원 2. 20,000 원 3. 30,000 원" << endl; 537. cout << "4. 40,000 원 5. 50,000 원 6. 100,000 원" << endl; 538. cout << "7. 직접 입력**\n**" << endl; 539. cout << "==================================================" << endl; 540. cout << "번호 입력 : "; 541. int aNum = -1; 542. cin >> aNum; 543. **if** (cin.fail() == true) { 544. atm->invalidKoreanDisplay(); 545. cin.clear(); 546. cin.ignore(100, '\n'); 547. **continue**; 548. } 549. **if** (aNum == 1) {inAmount = 10000; **break**;} 550. **else** **if** (aNum == 2) {inAmount = 20000; **break**;} 551. **else** **if** (aNum == 3) {inAmount = 30000; **break**;} 552. **else** **if** (aNum == 4) {inAmount = 40000; **break**;} 553. **else** **if** (aNum == 5) {inAmount = 50000; **break**;} 554. **else** **if** (aNum == 6) {inAmount = 100000; **break**;} 555. **else** **if** (aNum == 7) { 556. **while** (true) { 557. atm->mainKoreanDisplay(); 558. cout << " 희밍하는 출금 금액을 1만원 단위를 입력해주십시오**\n**"; 559. cout << "==================================================" << endl; 560. cout << "출금 금액 : "; 561. cin >> inAmount; 562. **if** (cin.fail() == true) { 563. atm->invalidKoreanDisplay(); 564. cin.clear(); 565. cin.ignore(100, '\n'); 566. **continue**; 567. } 568. **if** (inAmount < 0) atm->invalidKoreanDisplay(); 569. **else** **if** (inAmount == 0) { 570. atm->mainKoreanDisplay(); 571. cout << " 0 원을 출금할 수는 없습니다**\n**" << endl; 572. cout << "==================================================" << endl; 573. } 574. **else**{ 575. **if** ((inAmount % 10000 == 0) && (inAmount <= 500000)) **break**; 576. **if** (inAmount > 500000) { 577. atm->mainKoreanDisplay(); 578. cout << " 거래 1회 당 출금 가능한 액수를 초과하셨습니다**\n**" << endl; 579. cout << "==================================================" << endl; 580. } 581. **if** (inAmount % 10000 != 0) { 582. atm->mainKoreanDisplay(); 583. cout << " 반드시 1만원 단위로 입력하셔야 합니다**\n**" << endl; 584. cout << "==================================================" << endl; 585. } 586. } 587. } 588. **break**; 589. } **else** atm->invalidKoreanDisplay(); 590. } 591. atm->mainKoreanDisplay(); 592. Withdrawal(inAmount, 0); 593. cout << "==================================================" << endl; 594. }   598. } **else** **if** (transactionNum == 3) { *// Transfer* 599. atm->mainKoreanDisplay(); 600. cout << " 송금 서비스를 선택하셨습니다**\n**" << endl; 601. cout << " 1. 계좌 송금 (계좌 -> 계좌)" << endl; 602. cout << " 2. 현금 송금 (현금 -> 계좌)**\n**" << endl; 603. cout << "==================================================" << endl; 604. cout << "번호 입력 : "; 605. int transferNum = -1; 606. cin >> transferNum; 607. **if** (cin.fail() == true) { 608. atm->invalidKoreanDisplay(); 609. cin.clear(); 610. cin.ignore(100, '\n'); 611. **continue**; 612. } 614. **if** (transferNum == 1) { *// Account Transfer* 615. atm->mainKoreanDisplay(); 616. cout << " 계좌 송금 서비스를 선택하셨습니다**\n**" << endl; 617. cout << " 송금하실 금액을 입력해주십시오**\n**" << endl; 618. cout << "==================================================" << endl; 619. cout << "송금 금액 : "; 620. unsigned long long inAmount = -1; 621. cin >> inAmount; 622. **if** (cin.fail() == true) { 623. atm->invalidKoreanDisplay(); 624. cin.clear(); 625. cin.ignore(100, '\n'); 626. **continue**; 627. } 628. **if** (inAmount < 0) atm->invalidKoreanDisplay(); 629. **else** **if** (inAmount == 0) { 630. atm->mainKoreanDisplay(); 631. cout << " 0 원을 송금할 수는 없습니다**\n**" << endl; 632. cout << "==================================================" << endl; 633. } **else** { 634. atm->mainKoreanDisplay(); 635. cout << " 받으실 분의 계좌 번호를 입력해주십시오**\n**" << endl; 636. cout << "==================================================" << endl; 637. cout << "받으실 분 계좌 번호 : "; 638. string inDest; 639. cin >> inDest; 640. **if** (findAccount(inDest) == **nullptr**) { 641. atm->mainKoreanDisplay(); 642. cout << " 입력한 계좌번호가 존재하지 않습니다**\n**" << endl; 643. cout << "==================================================" << endl; 644. } **else** { 645. atm->mainKoreanDisplay(); 646. AccountTransfer(inAmount, findAccount(inDest)->findAccountOfBank(inDest), 0); 647. cout << "==================================================" << endl; 648. } 649. } 651. } **else** **if** (transferNum == 2) { *// Cash Transfer* 652. long long inAmount; 653. **while** (true) { 654. atm->mainKoreanDisplay(); 655. cout << " 송금하실 1만원권 지폐의 장 수를 입력해주세요**\n**" << endl; 656. cout << "==================================================" << endl; 657. cout << "1만원권 지폐 장 수 : "; 658. int numBill = -1; 659. cin >> numBill; 660. **if** (cin.fail() == true) { 661. atm->invalidKoreanDisplay(); 662. cin.clear(); 663. cin.ignore(100, '\n'); 664. **continue**; 665. } 666. **if** (numBill < 0) atm->invalidKoreanDisplay(); 667. **else** **if** (numBill == 0) { 668. atm->mainKoreanDisplay(); 669. cout << " 0 원을 송금할 수는 없습니다**\n**" << endl; 670. cout << "==================================================" << endl; 671. } **else** { 672. **if** (numBill <= 50) {inAmount = 10000 \* numBill; **break**;} 673. **else** { 674. atm->mainKoreanDisplay(); 675. cout << " 거래 1회 당 송금 가능한 장 수를 초과하셨습니다**\n**" << endl; 676. cout << "==================================================" << endl; 677. } 678. } 679. } 680. atm->mainKoreanDisplay(); 681. cout << " 받으실 분의 계좌 번호를 입력해주십시오**\n**" << endl; 682. cout << "==================================================" << endl; 683. cout << "받으실 분 계좌 번호 : "; 684. string inDest; 685. cin >> inDest; 686. **if** (findAccount(inDest) == **nullptr**) { 687. atm->mainKoreanDisplay(); 688. cout << " 입력한 계좌번호가 존재하지 않습니다**\n**" << endl; 689. cout << "==================================================" << endl; 690. } **else** { 691. atm->mainKoreanDisplay(); 692. AccountTransfer(inAmount, findAccount(inDest)->findAccountOfBank(inDest), 0); 693. cout << "==================================================" << endl; 694. } 696. } **else** { *// Exception* 697. atm->invalidKoreanDisplay(); 698. }  701. } **else** **if** (transactionNum == 4) { *// Transaction History* 702. atm->mainKoreanDisplay(); 703. cout << " 거래 내역 조회를 선택하셨습니다**\n**" << endl; 704. cout << " 해당 계좌의 거래내역은 다음과 같습니다**\n**" << endl; 705. vector<Transaction> temp = account->getTransactionHistoryOfAccount(); 706. **if** (temp.size() == 0) { 707. cout << "--------------------------------------------------**\n**" << endl; 708. cout << " 현재 잔액 : " << account->getFundInfo() << " 원**\n**" << endl; 709. cout << " 해당 계좌에는 거래 내역이 업습니다**\n**" << endl; 710. cout << "==================================================" << endl; 711. } **else** { 712. **for** (int i = 0; i < temp.size(); i++) { 713. cout << temp[i].getKoreanInformation() << endl; 714. } 715. cout << "**\n**현재 잔액 : " << account->getFundInfo() << " 원**\n**" << endl; 716. cout << "==================================================" << endl; 717. } 719. } **else** **if** (transactionNum == 5) { *// Session Exit* 720. sessionExitSignal = false; 721. }  724. **else** { *// Exception* 725. atm->invalidKoreanDisplay(); 726. } 727. } 728. atm->mainKoreanDisplay(); 729. cout << " 세션 종료" << endl; 730. cout << " 저희 ATM을 이용해주셔서 감사합니다**\n**" << endl; 731. **if** (transactionHistoryOfSession.size() == 0) { 732. cout << " 해당 세션에는 거래 내역이 업습니다**\n**" << endl; 733. } **else** { 734. cout << "==================================================" << endl; 735. cout << " 해당 세션 내 총 거래 내역" << endl; 736. cout << "==================================================" << endl; 737. atm->addTransaction(transactionHistoryOfSession); 738. **for** (int i = 0; i < transactionHistoryOfSession.size(); i++) { 739. cout << transactionHistoryOfSession[i].getKoreanInformation() << endl; 740. } 741. } 742. } 743. } 744. } 745. }; 746. */\*---------- English Session Class ----------\*/* 747. **class** **EnglishSession** : **public** Session { 748. **public**: 749. EnglishSession(ATM\* iatm) { 750. atm = iatm; 751. primarySignal = true; 752. authorizationCount = 0; 753. withdrawalCount = 0; 754. authorizationSignal = true; 755. bool validAccount = true; 756. string inputAccount; 757. atm->mainEnglishDisplay(); 758. cout << " PLEASE ENTER YOUR ACCOUNT NUMBER**\n**" << endl; 759. cout << "==================================================" << endl; 760. cout << "Account Number : "; 761. cin >> inputAccount; 763. **if** (findAccount(inputAccount) == **nullptr**) { 764. atm->mainEnglishDisplay(); 765. cout << " THE ACCOUNT NUMBER YOU ENTERED DOES NOT EXIST**\n**" << endl; 766. cout << "==================================================" << endl; 767. validAccount = false; 768. } 769. **else** { 770. Bank\* temp = findAccount(inputAccount); 771. **if** ( (atm->getPrimaryBankInfo()).compare(temp->getBankName()) == 0 ) { 772. account = temp->findAccountOfBank(inputAccount); 773. } **else** { 774. **if** (atm->getSingleInfo() == 0) { 775. atm->mainEnglishDisplay(); 776. cout << " YOU CAN'T USE OTHER BANK CARDS**\n**" << endl; 777. cout << "==================================================" << endl; 778. validAccount = false; 779. } **else** { 780. primarySignal = false; 781. account = temp->findAccountOfBank(inputAccount); 782. } 783. } 784. } 785. **if** (validAccount) { 786. **for** (int i = 1; i < 4; i++) { 787. string inputPassword; 788. atm->mainEnglishDisplay(); 789. cout << " PLEASE ENTER THE PASSWORD.**\n**" << endl; 790. cout << "==================================================" << endl; 791. cout << "Password : "; 792. cin >> inputPassword; 793. **if** (Authorization(inputPassword)) { 794. authorizationSignal = true; 795. **break**; 796. } 797. **else** { 798. authorizationSignal = false; 799. authorizationCount ++; 800. atm->mainEnglishDisplay(); 801. cout << " " << authorizationCount << " AUTHORIZATION FAIL**\n**" << endl; 802. cout << "==================================================" << endl; 803. } 804. } 805. **if** (authorizationSignal == false) { 806. atm->mainEnglishDisplay(); 807. cout << " 3 AUTHORIZATION FAIL, SESSION EXIT**\n**" << endl; 808. cout << "==================================================" << endl; 809. } 810. **else** { 811. bool sessionExitSignal = true; 812. **while** (sessionExitSignal) { 813. atm->mainEnglishDisplay(); 814. cout << " PLEASE CHOOSE THE SERVICE YOU WANT**\n**" << endl; 815. cout << " 1. DEPOSIT 2. WITHDRAWAL" << endl; 816. cout << " 3. TRANSFER 4. TRANSACTION HISTORY" << endl; 817. cout << " 5. SESSION EXIT**\n**" << endl; 818. cout << "==================================================" << endl; 819. cout << "Enter the number : "; 820. int transactionNum = -1; 821. cin >> transactionNum; 822. **if** (cin.fail() == true) { 823. atm->invalidEnglishDisplay(); 824. cin.clear(); 825. cin.ignore(100, '\n'); 826. **continue**; 827. }  830. **if** (transactionNum == 1) { *// Deposit* 831. atm->mainEnglishDisplay(); 832. cout << " You select the Deposit Transaction.**\n**" << endl; 833. cout << " SELECT CASH or CHECK**\n**" << endl; 834. cout << " 1. CASH DEPOSIT 2. CHECK DEPOSIT**\n**" << endl; 835. cout << "==================================================" << endl; 836. cout << "Enter the number : "; 837. int x = -1; 838. cin >> x; 839. **if** (cin.fail() == true) { 840. atm->invalidEnglishDisplay(); 841. cin.clear(); 842. cin.ignore(100, '\n'); 843. **continue**; 844. } 845. unsigned long long inAmount; 846. **if** (x == 1) { 847. **while** (true) { 848. atm->mainEnglishDisplay(); 849. cout << " PLEASE ENTER THE NUMBER OF 10,000 WON BILLS**\n**" << endl; 850. cout << "==================================================" << endl; 851. cout << "The Number of Bills : "; 852. int numBill = -1; 853. cin >> numBill; 854. **if** (cin.fail() == true) { 855. atm->invalidEnglishDisplay(); 856. cin.clear(); 857. cin.ignore(100, '\n'); 858. **continue**; 859. } 860. **if** (numBill < 0) atm->invalidEnglishDisplay(); 861. **else** **if** (numBill == 0) { 862. atm->mainEnglishDisplay(); 863. cout << " YOU CANNOT DEPOSIT 0 won**\n**" << endl; 864. cout << "==================================================" << endl; 865. } **else** { 866. **if** (numBill <= 50) {inAmount = 10000 \* numBill; **break**;} 867. **else** { 868. atm->mainEnglishDisplay(); 869. cout << " IT'S OVER THE LIMIT IN THE NUMBER OF CASH" << endl; 870. cout << " THAT CAN BE DEPOSITED PER TRANSACTION**\n**" << endl; 871. cout << "==================================================" << endl; 872. } 873. } 874. } 875. atm->mainEnglishDisplay(); 876. CashDeposit(inAmount, 1); 877. cout << "==================================================" << endl; 878. } **else** **if** (x == 2) { 879. **while** (true) { 880. atm->mainEnglishDisplay(); 881. cout << " PLEASE ENTER THE NUMBER OF 100,000 WON CHECKS**\n**" << endl; 882. cout << "==================================================" << endl; 883. cout << "The Number of Checks : "; 884. int numBill = -1; 885. cin >> numBill; 886. **if** (cin.fail() == true) { 887. atm->invalidEnglishDisplay(); 888. cin.clear(); 889. cin.ignore(100, '\n'); 890. **continue**; 891. } 892. **if** (numBill < 0) atm->invalidEnglishDisplay(); 893. **else** **if** (numBill == 0) { 894. atm->mainEnglishDisplay(); 895. cout << " YOU CANNOT DEPOSIT 0 won**\n**" << endl; 896. cout << "==================================================" << endl; 897. } **else** { 898. **if** (numBill <= 30) {inAmount = 100000 \* numBill; **break**;} 899. **else** { 900. atm->mainEnglishDisplay(); 901. cout << " IT'S OVER THE LIMIT IN THE NUMBER OF CHECK" << endl; 902. cout << " THAT CAN BE DEPOSITED PER TRANSACTION**\n**" << endl; 903. cout << "==================================================" << endl; 904. } 905. } 906. } 907. atm->mainEnglishDisplay(); 908. CheckDeposit(inAmount, 1); 909. cout << "==================================================" << endl; 910. } **else** atm->invalidEnglishDisplay();  913. } **else** **if** (transactionNum == 2) { *// Withdrawal* 914. **if** (withdrawalCount == 3) { 915. atm->mainEnglishDisplay(); 916. cout << " IT'S OVER THE LIMIT IN THE NUMBER OF" << endl; 917. cout << " WITHDRAWAL OF FUND PER SESSION**\n**" << endl; 918. cout << "==================================================" << endl; 919. } 920. **else** { 921. long long inAmount = -1; 922. **while** (true) { 923. atm->mainEnglishDisplay(); 924. cout << " You select the Withdrawal Transaction**\n**" << endl; 925. cout << " SELECT THE AMOUNT TO WITHDRAWAL**\n**" << endl; 926. cout << "1. 10,000 won 2. 20,000 won 3. 30,000 won" << endl; 927. cout << "4. 40,000 won 5. 50,000 won 6. 100,000 won" << endl; 928. cout << " 7. OTHER AMOUNT**\n**" << endl; 929. cout << "==================================================" << endl; 930. cout << "Enter the number : "; 931. int aNum = -1; 932. cin >> aNum; 933. **if** (cin.fail() == true) { 934. atm->invalidEnglishDisplay(); 935. cin.clear(); 936. cin.ignore(100, '\n'); 937. **continue**; 938. } 939. **if** (aNum == 1) {inAmount = 10000; **break**;} 940. **else** **if** (aNum == 2) {inAmount = 20000; **break**;} 941. **else** **if** (aNum == 3) {inAmount = 30000; **break**;} 942. **else** **if** (aNum == 4) {inAmount = 40000; **break**;} 943. **else** **if** (aNum == 5) {inAmount = 50000; **break**;} 944. **else** **if** (aNum == 6) {inAmount = 100000; **break**;} 945. **else** **if** (aNum == 7) { 946. **while** (true) { 947. atm->mainEnglishDisplay(); 948. cout << " ENTER THE AMOUNT IN UNITS OF 10,000 won**\n**"; 949. cout << "==================================================" << endl; 950. cout << "Amount : "; 951. cin >> inAmount; 952. **if** (cin.fail() == true) { 953. atm->invalidEnglishDisplay(); 954. cin.clear(); 955. cin.ignore(100, '\n'); 956. **continue**; 957. } 958. **if** (inAmount < 0) atm->invalidEnglishDisplay(); 959. **else** **if** (inAmount == 0) { 960. atm->mainEnglishDisplay(); 961. cout << " YOU CANNOT TRANSFER 0 won**\n**" << endl; 962. cout << "==================================================" << endl; 963. } **else** { 964. **if** ((inAmount % 10000 == 0) && (inAmount <= 500000)) **break**; 965. **if** (inAmount > 500000) { 966. atm->mainEnglishDisplay(); 967. cout << " IT'S OVER THE LIMIT IN THE AMOUNT OF" << endl; 968. cout << " WITHDRAWAL OF FUND PER TRANSACTION**\n**" << endl; 969. cout << "==================================================" << endl; 970. } 971. **if** (inAmount % 10000 != 0) { 972. atm->mainEnglishDisplay(); 973. cout << " YOU SHOULD ENTER THE AMOUNT" << endl; 974. cout << " IN UNITS OF 10,000 won**\n**" << endl; 975. cout << "==================================================" << endl; 976. } 977. } 978. } 979. **break**; 980. } **else** atm->invalidEnglishDisplay(); 981. } 982. atm->mainEnglishDisplay(); 983. Withdrawal(inAmount, 1); 984. cout << "==================================================" << endl; 985. }   989. } **else** **if** (transactionNum == 3) { *// Transfer* 990. atm->mainEnglishDisplay(); 991. cout << " You select the Transfer Transaction**\n**" << endl; 992. cout << " 1. Account Transfer (Account to Account)" << endl; 993. cout << " 2. Cash Transfer (Cash to Account)**\n**" << endl; 994. cout << "==================================================" << endl; 995. cout << "Enter the number : "; 996. int transferNum = -1; 997. cin >> transferNum; 998. **if** (cin.fail() == true) { 999. atm->invalidEnglishDisplay(); 1000. cin.clear(); 1001. cin.ignore(100, '\n'); 1002. **continue**; 1003. } 1005. **if** (transferNum == 1) { *// Account Transfer* 1006. atm->mainEnglishDisplay(); 1007. cout << " You select the Account Transfer Transaction**\n**" << endl; 1008. cout << " PLEASE ENTER THE AMOUNT TO TRANSFER**\n**" << endl; 1009. cout << "==================================================" << endl; 1010. cout << "Amount : "; 1011. long long inAmount = -1; 1012. cin >> inAmount; 1013. **if** (cin.fail() == true) { 1014. atm->invalidEnglishDisplay(); 1015. cin.clear(); 1016. cin.ignore(100, '\n'); 1017. **continue**; 1018. } 1019. **if** (inAmount < 0) atm->invalidEnglishDisplay(); 1020. **else** **if** (inAmount == 0) { 1021. atm->mainEnglishDisplay(); 1022. cout << " YOU CANNOT TRANSFER 0 won**\n**" << endl; 1023. cout << "==================================================" << endl; 1024. } **else** { 1025. atm->mainEnglishDisplay(); 1026. cout << " PLEASE ENTER THE AMOUNT TO TRANSFER**\n**" << endl; 1027. cout << "==================================================" << endl; 1028. cout << "Destination Account Number : "; 1029. string inDest; 1030. cin >> inDest; 1031. **if** (findAccount(inDest) == **nullptr**) { 1032. atm->mainEnglishDisplay(); 1033. cout << " THE ACCOUNT NUMBER YOU ENTERED DOES NOT EXIST**\n**" << endl; 1034. cout << "==================================================" << endl; 1035. } **else** { 1036. atm->mainEnglishDisplay(); 1037. AccountTransfer(inAmount, findAccount(inDest)->findAccountOfBank(inDest), 1); 1038. cout << "==================================================" << endl; 1039. } 1040. } 1042. } **else** **if** (transferNum == 2) { *// Cash Transfer* 1043. unsigned long long inAmount; 1044. **while** (true) { 1045. atm->mainEnglishDisplay(); 1046. cout << " PLEASE ENTER THE NUMBER OF 10,000 WON BILLS**\n**" << endl; 1047. cout << "==================================================" << endl; 1048. cout << "The Number of Bills : "; 1049. int numBill = -1; 1050. cin >> numBill; 1051. **if** (cin.fail() == true) { 1052. atm->invalidEnglishDisplay(); 1053. cin.clear(); 1054. cin.ignore(100, '\n'); 1055. **continue**; 1056. } 1057. **if** (numBill < 0) atm->invalidEnglishDisplay(); 1058. **else** **if** (numBill == 0) { 1059. atm->mainEnglishDisplay(); 1060. cout << " YOU CANNOT TRANSFER 0 won**\n**" << endl; 1061. cout << "==================================================" << endl; 1062. } **else** { 1063. **if** (numBill <= 50) {inAmount = 10000 \* numBill; **break**;} 1064. **else** { 1065. atm->mainEnglishDisplay(); 1066. cout << " IT'S OVER THE LIMIT IN THE NUMBER OF CASH" << endl; 1067. cout << " THAT CAN BE TRANSFERED PER TRANSACTION**\n**" << endl; 1068. cout << "==================================================" << endl; 1069. } 1070. } 1071. } 1072. atm->mainEnglishDisplay(); 1073. cout << " PLEASE ENTER THE ACCOUNT TO TRANSFER**\n**" << endl; 1074. cout << "==================================================" << endl; 1075. cout << "Destination Account Number : "; 1076. string inDest; 1077. cin >> inDest; 1078. **if** (findAccount(inDest) == **nullptr**) { 1079. atm->mainEnglishDisplay(); 1080. cout << " THE ACCOUNT NUMBER YOU ENTERED DOES NOT EXIST**\n**" << endl; 1081. cout << "==================================================" << endl; 1082. } **else** { 1083. atm->mainEnglishDisplay(); 1084. AccountTransfer(inAmount, findAccount(inDest)->findAccountOfBank(inDest), 1); 1085. cout << "==================================================" << endl; 1086. } 1088. } **else** { *// Exception* 1089. atm->invalidEnglishDisplay(); 1090. }  1093. } **else** **if** (transactionNum == 4) { *// Transaction History* 1094. atm->mainEnglishDisplay(); 1095. cout << " You select the Transaction History**\n**" << endl; 1096. cout << " THE TRANSACTION DETAILS OF" << endl; 1097. cout << " THE ACCOUNT ARE AS FOLLOWS**\n**" << endl; 1098. vector<Transaction> temp =account->getTransactionHistoryOfAccount(); 1099. **if** (temp.size() == 0) { 1100. cout << "==================================================**\n**" << endl; 1101. cout << "CURRENT BALANCE : " << account->getFundInfo() << " won**\n**" << endl; 1102. cout << "THIS ACCOUNT DOESN'T HAVE ANY TRANSACTION HISTORY**\n**" << endl; 1103. } **else** { 1104. **for** (int i = 0; i < temp.size(); i++) { 1105. cout << temp[i].getEnglishInformation() << endl; 1106. } 1107. cout << "**\n**CURRENT BALANCE : " << account->getFundInfo() << " won**\n**" << endl; 1108. cout << "==================================================" << endl; 1109. } 1111. } **else** **if** (transactionNum == 5) { *// Session Exit* 1112. sessionExitSignal = false; 1113. }  1116. **else** { *// Exception* 1117. atm->invalidEnglishDisplay(); 1118. } 1119. } 1120. atm->mainEnglishDisplay(); 1121. cout << " SESSION END" << endl; 1122. cout << " THANK YOU FOR USING OUR ATM**\n**" << endl; 1123. **if** (transactionHistoryOfSession.size() == 0) { 1124. cout << "THIS SESSION DOESN'T HAVE ANY TRANSACTION HISTORY**\n**" << endl; 1125. } **else** { 1126. cout << "==================================================" << endl; 1127. cout << " TRANSACTION HISTORY OF SESSION" << endl; 1128. cout << "==================================================" << endl; 1129. atm->addTransaction(transactionHistoryOfSession); 1130. **for** (int i = 0; i < transactionHistoryOfSession.size(); i++) { 1131. cout << transactionHistoryOfSession[i].getEnglishInformation() << endl; 1132. } 1133. } 1134. } 1135. } 1136. } 1137. }; 1138. */\*---------------- Method of ATM Class ----------------\*/* 1139. void ATM::startEnglishAdminSession() { 1140. **this**->mainEnglishDisplay(); 1141. cout << " You select the Admin Menu" << endl; 1142. cout << " PLEASE ENTER THE ADMIN CARD NUMBER**\n**" << endl; 1143. cout << "==================================================" << endl; 1144. cout << "Admin Card Number : "; 1145. string inAdmin; 1146. cin >> inAdmin; 1147. **if** (inAdmin.compare(AdminNum) == 0) { 1148. **while** (true) { 1149. **this**->mainEnglishDisplay(); 1150. cout << " SELECT THE ADMIN SERVICE**\n**" << endl; 1151. cout << " 1. TRANSACTION HISTORY OF ATM" << endl; 1152. cout << " 2. ADMIN SESSION EXIT**\n**" << endl; 1153. cout << "==================================================" << endl; 1154. cout << "Enter the number : "; 1155. int num = -1; 1156. cin >> num; 1157. **if** (cin.fail() == true) { 1158. invalidEnglishDisplay(); 1159. cin.clear(); 1160. cin.ignore(100, '\n'); 1161. **continue**; 1162. } 1163. **if** (num == 1) { 1164. **this**->mainEnglishDisplay(); 1165. **if** (transactionHistoryOfATM.size() == 0) { 1166. cout << " THIS ATM HAS NO TRANSACTION HISTORY" << endl; 1167. } **else** { 1168. ofstream fout; 1169. fout.open("Transaction History of ATM.txt"); 1170. **if** (!fout) cout << " File Error" << endl; 1171. **else** { 1172. fout << "ATM Serial No. : " << **this**->getSerialNum() << endl; 1173. **for** (int i = 0; i < transactionHistoryOfATM.size(); i++) { 1174. **for** (int j = 0; j < transactionHistoryOfATM[i].size(); j++) { 1175. fout << transactionHistoryOfATM[i][j].getEnglishInformation(); 1176. fout << "**\n**"; 1177. } 1178. } 1179. } 1180. cout << "==================================================" << endl; 1181. cout << " TRANSACTION HISTORY OF THIS ATM" << endl; 1182. cout << "==================================================" << endl; 1183. **for** (int i = 0; i < transactionHistoryOfATM.size(); i++) { 1184. **for** (int j = 0; j < transactionHistoryOfATM[i].size(); j++) { 1185. cout << transactionHistoryOfATM[i][j].getEnglishInformation() << endl; 1186. } 1187. } 1188. } 1189. } **else** **if** (num == 2) { 1190. **this**->mainEnglishDisplay(); 1191. cout << " ADMIN SESSION END" << endl; 1192. **break**; 1193. } **else** **this**->invalidEnglishDisplay(); 1194. } 1195. } **else** **this**->invalidEnglishDisplay(); 1196. } 1197. void ATM::startKoreanAdminSession() { 1198. **this**->mainKoreanDisplay(); 1199. cout << " 관리자 모드를 선택하셨습니다" << endl; 1200. cout << " 관리자 카드 번호를 입력해주십시오**\n**" << endl; 1201. cout << "==================================================" << endl; 1202. cout << "관리자 카드 번호 : "; 1203. string inAdmin; 1204. cin >> inAdmin; 1205. **if** (inAdmin.compare(AdminNum) == 0) { 1206. **while** (true) { 1207. **this**->mainKoreanDisplay(); 1208. cout << " 이용하실 서비스를 선택해주십시오**\n**" << endl; 1209. cout << " 1. ATM 총 거래 내역" << endl; 1210. cout << " 2. 관리자 모드 종료**\n**" << endl; 1211. cout << "==================================================" << endl; 1212. cout << "번호 입력 : "; 1213. int num = -1; 1214. cin >> num; 1215. **if** (cin.fail() == true) { 1216. invalidKoreanDisplay(); 1217. cin.clear(); 1218. cin.ignore(100, '\n'); 1219. **continue**; 1220. } 1221. **if** (num == 1) { 1222. **this**->mainKoreanDisplay(); 1223. **if** (transactionHistoryOfATM.size() == 0) { 1224. cout << " 해당 ATM 기기에는 거래 내역이 없습니다" << endl; 1225. } **else** { 1226. ofstream fout; 1227. fout.open("Transaction History of ATM.txt"); 1228. **if** (!fout) cout << " 출력 파일 오류" << endl; 1229. **else** { 1230. fout << "ATM 고유식별 번호 : " << **this**->getSerialNum() << endl; 1231. **for** (int i = 0; i < transactionHistoryOfATM.size(); i++) { 1232. **for** (int j = 0; j < transactionHistoryOfATM[i].size(); j++) { 1233. fout << transactionHistoryOfATM[i][j].getKoreanInformation(); 1234. fout << "**\n**"; 1235. } 1236. } 1237. } 1238. cout << "==================================================" << endl; 1239. cout << " 해당 ATM 총 거래 내역" << endl; 1240. cout << "==================================================" << endl; 1241. **for** (int i = 0; i < transactionHistoryOfATM.size(); i++) { 1242. **for** (int j = 0; j < transactionHistoryOfATM[i].size(); j++) { 1243. cout << transactionHistoryOfATM[i][j].getKoreanInformation() << endl; 1244. } 1245. } 1246. } 1247. } **else** **if** (num == 2) { 1248. **this**->mainKoreanDisplay(); 1249. cout << " 관리자 모드 종료" << endl; 1250. **break**; 1251. } **else** **this**->invalidKoreanDisplay(); 1252. } 1253. } **else** **this**->invalidKoreanDisplay(); 1254. } 1255. */\*---------------- Children Classes of ATM Class ----------------\*/* 1256. */\*---------- Single Bank ATM Class ----------\*/* 1257. **class** **SingleBankATM** : **public** ATM { 1258. **public**: 1259. SingleBankATM(string priName, string amount) { 1260. serialNum = atmCnt++; 1261. primaryBank = findBank(priName); 1262. cashAmount = stoull(amount); 1263. SingleOrMulti = 0; 1264. } 1265. void startSession() {} 1266. string getClassName() {**return** "Single";} 1267. }; 1268. */\*---------- Multi Bank ATM Class -----------\*/* 1269. **class** **MultiBankATM** : **public** ATM { 1270. **public**: 1271. MultiBankATM(string priName, string amount) { 1272. serialNum = atmCnt++; 1273. primaryBank = findBank(priName); 1274. cashAmount = stoull(amount); 1275. SingleOrMulti = 1; 1276. } 1277. void startSession() {} 1278. string getClassName() {**return** "Multi";} 1279. }; 1280. */\*---------- Unilingual ATM Class -----------\*/* 1281. **class** **UnilingualATM** : **public** ATM { 1282. **public**: 1283. UnilingualATM(ATM\* atm, string admin) { 1284. serialNum = atm->getSerialNum(); 1285. primaryBank = atm->getPrimaryBank(); 1286. cashAmount = atm->getCashAmount(); 1287. SingleOrMulti = atm->getSingleInfo(); 1288. AdminNum = admin; 1289. } 1290. string getClassName() {**return** "Unilingual";} 1291. void startSession() { 1292. **while** (true) { 1293. **this**->mainEnglishDisplay(); 1294. cout << " WELCOME **\n**" << endl; 1295. cout << " MENU**\n**" << endl; 1296. cout << " 1. TRANSACTION 2. ADMIN" << endl; 1297. cout << " 3. Go to HOME**\n**" << endl; 1298. cout << "==================================================" << endl; 1299. cout << "번호 입력 : "; 1300. int languageNum = -1; 1301. cin >> languageNum; 1302. **if** (cin.fail() == true) { 1303. invalidEnglishDisplay(); 1304. cin.clear(); 1305. cin.ignore(100, '\n'); 1306. **continue**; 1307. } 1308. **if** (languageNum == 1) { 1309. cout << "**\n**--------------------------------------------------**\n**" << endl; 1310. EnglishSession newSession(**this**); 1311. session = newSession; 1312. } **else** **if** (languageNum == 2) { 1313. startEnglishAdminSession(); 1314. } **else** **if** (languageNum == 3) { 1315. **break**; 1316. } **else** **this**->invalidEnglishDisplay(); 1317. } 1318. } 1319. }; 1320. */\*----------- Bilingual ATM Class -----------\*/* 1321. **class** **BilingualATM** : **public** ATM { 1322. **public**: 1323. BilingualATM(ATM\* atm, string admin) { 1324. serialNum = atm->getSerialNum(); 1325. primaryBank = atm->getPrimaryBank(); 1326. cashAmount = atm->getCashAmount(); 1327. SingleOrMulti = atm->getSingleInfo(); 1328. AdminNum = admin; 1329. } 1330. string getClassName() {**return** "Bilingual";} 1331. void startSession() { 1332. int sign = startEnglishSession(); 1333. **if** (sign != 0) { 1334. **do** { 1335. **if** (sign == 1) sign = startKoreanSession(); 1336. **else** sign = startEnglishSession(); 1337. } **while** (sign != 0); 1338. } 1339. } 1340. int startEnglishSession() { 1341. **while** (true) { 1342. **this**->mainEnglishDisplay(); 1343. cout << " WELCOME **\n**" << endl; 1344. cout << " MENU**\n**" << endl; 1345. cout << " 1. TRANSACTION 2. 한국어(KOREAN)" << endl; 1346. cout << " 3. ADMIN 4. Go to HOME**\n**" << endl; 1347. cout << "==================================================" << endl; 1348. cout << "Enter the number : "; 1349. int languageNum = -1; 1350. cin >> languageNum; 1351. **if** (cin.fail() == true) { 1352. invalidEnglishDisplay(); 1353. cin.clear(); 1354. cin.ignore(100, '\n'); 1355. **continue**; 1356. } 1357. **if** (languageNum == 1) { 1358. EnglishSession newSession(**this**); 1359. session = newSession; 1360. } **else** **if** (languageNum == 2) { 1361. **return** 1; 1362. } **else** **if** (languageNum == 3) { 1363. startEnglishAdminSession(); 1364. } **else** **if** (languageNum == 4) { 1365. **return** 0; 1366. } **else** **this**->invalidEnglishDisplay(); 1367. } 1368. } 1369. int startKoreanSession() { 1370. **while** (true) { 1371. **this**->mainKoreanDisplay(); 1372. cout << " 환영합니다 **\n**" << endl; 1373. cout << " 메뉴**\n**" << endl; 1374. cout << " 1. 서비스 이용 2. ENGLISH(영어)" << endl; 1375. cout << " 3. 관리자 모드 4. ATM 나가기**\n**" << endl; 1376. cout << "==================================================" << endl; 1377. cout << "번호 입력 : "; 1378. int languageNum = -1; 1379. cin >> languageNum; 1380. **if** (cin.fail() == true) { 1381. invalidKoreanDisplay(); 1382. cin.clear(); 1383. cin.ignore(100, '\n'); 1384. **continue**; 1385. } 1386. **if** (languageNum == 1) { 1387. cout << "**\n**--------------------------------------------------**\n**" << endl; 1388. KoreanSession newSession(**this**); 1389. session = newSession; 1390. } **else** **if** (languageNum == 2) { 1391. cout << "**\n**--------------------------------------------------**\n**" << endl; 1392. **return** 2; 1393. } **else** **if** (languageNum == 3) { 1394. startKoreanAdminSession(); 1395. } **else** **if** (languageNum == 4) { 1396. **return** 0; 1397. } **else** **this**->invalidKoreanDisplay(); 1398. } 1399. } 1400. }; 1401. */\*------------------------------------- FILE Read Function ------------------------------------\*/* 1402. vector<string> split(string input, char delimeter) { 1403. vector<string> result; 1404. stringstream in(input); 1405. string temp; 1407. **while** (getline(in, temp, delimeter)) { 1408. result.push\_back(temp); 1409. } 1411. **return** result; 1412. } 1413. void readAtmData(ifstream& fin) { 1414. **if** (!fin) { 1415. cout << "해당 파일이 존재하지 않습니다." << endl; 1416. exit(6); 1417. } **else** { 1418. **while** (!fin.eof()) { 1419. */\** 1420. *splitted[0] : 은행 이름* 1421. *splitted[1] : "Single" or "Multi"* 1422. *splitted[2] : "Bi" or "Uni" (Bilingual or Unilingual)* 1423. *splitted[3] : ATM 기기 내 잔고* 1424. *splitted[4] : Admin Card Number (8글자)* 1425. *\*/* 1426. string str; 1427. getline(fin, str); 1428. vector<string> splitted = split(str, ' '); 1429. ATM\* newAtm; 1430. **if** (splitted[1].compare("Single") == 0) { 1431. newAtm = **new** SingleBankATM(splitted[0], splitted[3]); 1432. } **else** **if** (splitted[1].compare("Multi") == 0) { 1433. newAtm = **new** MultiBankATM(splitted[0], splitted[3]); 1434. } **else** { 1435. cout << "입력 파일 오류" << endl; 1436. exit(7); 1437. } 1438. **if** (splitted[2].compare("Bi") == 0) { 1439. newAtm = **new** BilingualATM(newAtm, splitted[4]); 1440. } **else** **if** (splitted[2].compare("Uni") == 0) { 1441. newAtm = **new** UnilingualATM(newAtm, splitted[4]); 1442. } **else** { 1443. cout << "입력 파일 오류" << endl; 1444. exit(8); 1445. } 1446. atmData.push\_back(newAtm); 1447. } 1448. } 1449. } 1450. void readBankData(ifstream& fin) { 1451. **if** (!fin) { 1452. cout << "해당 파일이 존재하지 않습니다." << endl; 1453. exit(9); 1454. } **else** { 1455. **while** (!fin.eof()) { 1456. *// str : 은행 이름* 1457. string str; 1458. getline(fin, str); 1459. Bank newBank(str); 1460. bankData.push\_back(newBank); 1461. } 1462. } 1463. } 1464. void readAccountData(ifstream& fin) { 1465. **if** (!fin) { 1466. cout << "해당 파일이 존재하지 않습니다." << endl; 1467. exit(10); 1468. } **else** { 1469. **while** (!fin.eof()) { 1470. */\** 1471. *splitted[0] : 은행 이름* 1472. *splitted[1] : 계좌번호 (8글자)* 1473. *splitted[2] : 비밀번호 (4글자)* 1474. *splitted[3] : 이름* 1475. *splitted[4] : 계좌 잔고* 1476. *\*/* 1477. string str; 1478. getline(fin, str); 1479. vector<string> splitted = split(str, ' '); 1480. Account newAccount(splitted[1], splitted[2], splitted[3], splitted[4]); 1481. findBank(splitted[0])->addAccount(newAccount); 1482. } 1483. } 1484. } 1485. */\*--------------------------------------- Main Function ---------------------------------------\*/* 1486. int main(int argc, char\* argv[]) { 1487. */\** 1489. *argc 는 인자 개수* 1490. *argv 는 인자 리스트 (문자열로 저장됨)* 1491. *for 구문을 통해 ATM, Bank, Account 데이터를 읽어들일 거임* 1492. *.txt 파일을 읽어들일 거며 .txt 파일은 ATM, Bank, Account 총 3개* 1493. *argv[0] : 프로그램명* 1494. *argv[1] ~ argv[3] : Bank.txt, ATM.txt, Account.txt* 1495. *-> 따라서 argc는 4* 1496. *\*/*  1499. */\** 1500. *자신의 환경에 따라 두 가지 조건 중 자신이 해당되는 것을 골라 그렇지 않은 케이스의 코드 주석 처리하여 사용하시오.* 1501. *1. Xcode(Mac OS) 이거나 VScode(Window OS) 에서 코드를 실행할 경우 --->>> Case 1 사용* 1502. *2. Terminal 에서 코드를 직접 컴파일하여 사용할 경우 --->>> Case 2 사용* 1503. *\*/* 1505. */\*------------------------------------------------------- Case 1 Start -------------------------------------------------------\*/* 1506. *// 해당 파일이 동일한 디렉토리에 있어야 됨.* 1507. */\** 1508. *Xcode의 경우, "Product" -> "Scheme" -> "Edit Scheme" -> "Run" -> "Options"* 1509. *-> "Working Directory"에서 "Use custom working directory" 체크* 1510. *\*/* 1511. ifstream f2("Bank.txt"); 1512. readBankData(f2); 1513. ifstream f1("ATM.txt"); 1514. readAtmData(f1); 1515. ifstream f3("Account.txt"); 1516. readAccountData(f3); 1517. */\*-------------------------------------------------------- Case 1 End --------------------------------------------------------\*/*  1520. */\*------------------------------------------------------- Case 2 Start -------------------------------------------------------\*/* 1521. *// // 해당 파일이 동일한 디렉토리에 있어야 됨.* 1522. *// int atmArgCount = 0;* 1523. *// int bankArgCount = 0;* 1524. *// int accountArgCount = 0;* 1525. *//* 1526. *// if (argc != 4) { // Argument 수가 맞지 않을때* 1527. *// cout << "Argument 부족" << endl;* 1528. *// return 1;* 1529. *// } else { // 정상 실행* 1530. *// for (int i = 1; i < argc; i++) {* 1531. *// if (strncmp(argv[i], "ATM.txt", 7) == 0) { // Argument 가 ATM.txt 일 경우* 1532. *// if (atmArgCount != 0) { // ATM argument 가 이미 한 번 나왔는데 또 나왔을 경우, Error 발생* 1533. *// cout << "Too many ATM arguments..." << endl;* 1534. *// return 2;* 1535. *// } else { // 정상 실행* 1536. *// ifstream fin(argv[i]); // 파일 read* 1537. *// readAtmData(fin);* 1538. *// atmArgCount ++;* 1539. *// }* 1540. *// } else if (strncmp(argv[i], "Bank.txt", 8) == 0) { // Argument 가 Bank.txt 일 경우* 1541. *// if (bankArgCount != 0) { // Bank argument 가 이미 한 번 나왔는데 또 나왔을 경우, Error 발생* 1542. *// cout << "Too many Bank arguments..." << endl;* 1543. *// return 3;* 1544. *// } else { // 정상 실행* 1545. *// ifstream fin(argv[i]); // 파일 read* 1546. *// readBankData(fin);* 1547. *// bankArgCount ++;* 1548. *// }* 1549. *// } else if (strncmp(argv[i], "Account.txt", 11) == 0) { // Argument 가 Account.txt 일 경우* 1550. *// if (accountArgCount != 0) { // Account argument 가 이미 한 번 나왔는데 또 나왔을 경우, Error 발생* 1551. *// cout << "Too many Account arguments..." << endl;* 1552. *// return 4;* 1553. *// } else { // 정상 실행* 1554. *// ifstream fin(argv[i]); // 파일 read* 1555. *// readAccountData(fin);* 1556. *// accountArgCount ++;* 1557. *// }* 1558. *// } else { // Exception* 1559. *// cout << argv[i] << "의 파일명이 invalid 합니다..." << endl;* 1560. *// return 5;* 1561. *// }* 1562. *// }* 1563. *// }* 1564. */\*-------------------------------------------------------- Case 2 End --------------------------------------------------------\*/* 1566. bool programEndSignal = true; 1568. **while** (programEndSignal) { 1569. cout << "Select the ATM" << endl; 1570. cout << "이용하실 ATM을 선택해주십시오**\n**" << endl; 1571. **for** (int i = 0; i < atmData.size(); i++) { 1572. cout << right << setfill('0') << setw(2) << i+1 << ". " << left << setfill(' ') << setw(8) << atmData[i]->getPrimaryBankInfo()<< " "; 1573. **if** (atmData[i]->getSingleInfo() == 0) { 1574. cout << "Single "; 1575. } **else** cout << "Multi "; 1576. cout << left << setw(10) << atmData[i]->getClassName() << " ATM" <<endl; 1577. } 1578. cout << atmData.size()+1 << ". Program Exit 프로그램 종료**\n**" << endl; 1579. cout << "Enter the Number (번호 입력) : "; 1580. int choiceAtm = -1; 1581. cin >> choiceAtm; 1582. **if** (cin.fail() == true) { 1583. cout << "**\n**==================================================" << endl; 1584. cout << " It's an INVALID number." << endl; 1585. cout << " 유효하지 않은 번호입니다." << endl; 1586. cout << "==================================================**\n**" << endl; 1587. cin.clear(); 1588. cin.ignore(100, '\n'); 1589. **continue**; 1590. } 1591. **if** (choiceAtm == atmData.size()+1) { 1592. programEndSignal = false; 1593. } 1594. **else** **if** ((choiceAtm > 0) && (choiceAtm < atmData.size())) { 1595. atmData[choiceAtm-1]->startSession(); 1596. } **else** { 1597. cout << "**\n**==================================================" << endl; 1598. cout << " It's an INVALID number." << endl; 1599. cout << " 유효하지 않은 번호입니다." << endl; 1600. cout << "==================================================**\n**" << endl; 1601. } 1602. } 1604. **return** 0; 1605. } |

# 7. Member student contribution table and note

|  |  |  |
| --- | --- | --- |
| 학번 | 이름 | 기여도 |
| 201511047 | Kim, Jong-Soo | % |
| 201611030 | Kim, San-Ha | % |
| 201911012 | Kong, Young-Jae | % |
| 201911088 | An, Sang-Jun | % |