Assignment one report

Team member:

Zimo Wang

Yewin Shu

GuanLin Lu

Yijia Zhang

Introduction:

In this assignment, frontend of the banking system prototype is developed with python by using a single class structure and numerous method functions. Each function provides different input possibilities and eliminates illegal user selections and generating a corresponding output for different cases.

Code structure:

The user class is generated as **"self"** object to represent a simulated customer. All the method and attributes are stored in the initiation function, which can be accessed through self object. And the system will initiate the sequence by introducing the user to the userLogin() method:

```
class user(object):
    def __init__(self):
        self.function = ""
        self.mode = None
        self.login = False
        self.logout = False
        self.acctNum = None
        self.acctName = None
        self.deposited = 0.00
        self.withdrawn = 0.00
        self.transferred = 0.00
        self.balance = 0.00
```

```
if __name__ == '__main__':
    myUser = user()
    myUser.userLogin()
```

The userLogin() method ask user to choose the option from the following list, but only login is the valid choice. Anything other than login will result in an error message, and the function will run recursively until the user choose the right option:

```
Welcome to use our bank system please select the following option
1. Login to an account(login):
2. Create an account (createacct)
3. Delete an account (deleteacct)
4. Deposit money into an account(deposit)
5. With draw money from an account(withdraw)
6. Transfer money from one account to the other (transfer)
7. Logging out(logout)
please enter your choice: createacct
Can not operate before login
Welcome to use our bank system please select the following option
1. Login to an account(login):
2. Create an account (createacct)
3. Delete an account (deleteacct)
4. Deposit money into an account(deposit)
5. With draw money from an account(withdraw)
6. Transfer money from one account to the other (transfer)
7. Logging out(logout)
please enter your choice: asdsadasda
Can not operate before login
Welcome to use our bank system please select the following option
1. Login to an account(login):
2. Create an account (createacct)
3. Delete an account (deleteacct)
4. Deposit money into an account(deposit)
5. With draw money from an account(withdraw)
6. Transfer money from one account to the other (transfer)
7. Logging out(logout)
please enter your choice: login
Select mode:
```

The userMode() method ask for the desired mode for user with two possible options: **agent** , **machine**. But system itself will have three possible outcomes: **agent, machine, and invalid inputs**. If user choose agent mode or machine mode. On the other hand, error message will pop out.

```
def userMode(self):
    self.mode = input("Select mode: ")
    if self.mode != "machine" and self.mode != "agent":
        print("error")
        return self.userLogin()
    else:
        self.login = True
        self.chooseFunction()
```

The chooseFunction() method ask for the user to select the one of the transaction code it provided. User cannot login when login state equals true. If user log out the system, it will generate a transaction file. Any other option will send user to their selected transaction prompt area.

- createacct

The createacct() function create an account and append it to the current valid account file. Function first check if the user login as agent mode. If the mode of the user is not agent, it will generate error messages and exit the system. On the other hand, if the mode is valid, the system will ask for account number. Moreover, it will check the validity of the account number and if there is any duplication with the existing account. After all the error checking procedure, the new account has been append to the validAccount file.

```
def createacct(self):
    if self.mode != "agent":
        print("error1")
        return 0
    else:
        acctNum = input("Please enter the account number: ")

        if acctNum not in self.fileOpen("ValidAccounts.txt") and self.checkAccNum(int(acctNum)):
            print("Valid number")
            self.acctNum = acctNum
            self.acctName = input("Please enter the account name: ")
            print("Successfully create the account")
            self.fileAppend("ValidAccounts.txt", self.acctNum)
            content = "NEW " + self.acctNum + ' ' + '$0.00' + ' 0000000 ' + self.acctName + '\n'
            self.writeSummary("transactionSummaryFile.txt", content)
            return self.chooseFunction()
        else:
            print("Invalid ACCOUNT NUMBER")
            return self.createacct()
```

-deleteacct

The deleteacct() function deletes an existed account in term of the input account number. It first check through the validAccount file to see the existent of the account. After this procedure, system ask for account name. If the name is valid, then account will be removed from the account file. If the name is invalid, it will ask user again for a valid account name.

```
def deleteacct(self):
    accNumber = input("Please enter the number that you want to delete: ")
    if self.checkAccNum(int(accNumber)) == True:
        list = self.fileOpen("ValidAccounts.txt")
        print(list)
        if accNumber in self.fileOpen("ValidAccounts.txt"):
            accountName = input("Please enter the name of that account: ")
            if self.checkAccName(accountName) == True:
                for i in range(len(list) - 1):
                    if accNumber == list[i]:
                        list.remove(list[i])
                tempFile = open("ValidAccounts.txt", "w")
                for line in list:
                    tempFile.write(line + "\n")
                tempFile.close()
                print("Account successfully deleted")
                content = "DEL " + str(accNumber) + ' $0.00 ' + ' 0000000 ' + accountName + '\n'
                self.writeSummary("transactionSummaryFile.txt", content)
                return self.chooseFunction()
            else:
                return self.deleteacct()
    else:
        return self.deleteacct()
```

-deposit

The deposit() function makes deposit for an account. System ask for Account name with constraint: deposits above $2,000 should be rejected in ATM mode(max $999,999.99 in agent mode). For ATM, the daily deposit limit is $5,000 per account. At the end it will write the deposit amount into the corresponding account.

```
def deposit(self):

    # Enter deposit transaction
    self.function = "deposit"
    accountNumber = int(input("Enter acc num: "))
    while self.checkAccNum(accountNumber) == False:
        print("deposit acc num error")
        accountNumber = int(input("Enter acc num: "))
        if str(accountNumber) in self.fileOpen("ValidAccounts.txt"):
            break

    amount = input("Enter amount to be deposited: ")
    while self.checkAmount(amount) == False:
        print("deposit amount error")
        amount = input("Enter amount to be deposited: ")

    if self.checkAccNum(accountNumber) and self.checkAmount(amount):
        self.deposited += float("{:.2f}".format(int(amount) / 100))
        print(str(int(self.deposited * 100)))
        if self.deposited <= 5000.0:
            print(str(self.deposited) + "deposited")
            self.balance += float("{:.2f}".format(int(amount) / 100))
            content = "DEP " + str(accountNumber) + ' ' + '$' + str(
                float("{:.2f}".format(int(amount) / 100))) + ' 0000000 ' + '***' + '\n'
            self.writeSummary("transactionSummaryFile.txt", content)
        else:
            print("Exceeded daily limit!")

    return self.chooseFunction()
```

-withdraw

Again, the function will ask for account number and account name, and check for account balance. Before with draw money from the account with some constraints.

```
def withdraw(self):

    self.function = "withdraw"
    accountNumber = int(input("Enter acc num: "))
    while self.checkAccNum(accountNumber) == False:
        print("withdraw acc num error")
        accountNumber = int(input("Enter acc num: "))

    amount = input("Enter amount to be withdrawn: ")
    while self.checkAmount(amount) == False:
        print("withdraw amount error")
        amount = input("Enter amount to be withdrawn: ")
    if self.checkAccNum(accountNumber) and self.checkAmount(amount):
        self.withdrawn += float("{:.2f}".format(int(amount) / 100))
        self.balance -= self.withdrawn
        print(self.balance)
    else:
        self.logout = True
    return self.chooseFunction()
```

-transfer

The transfer() function will ask for an from account number, an to account number and the amount to be transferred. Then system will check the validity of the account numbers and the amount to be transferred.

```
def transfer(self):
    self.function = "transfer"

    fromAcc = int(input("Enter the from acc num: "))
    while self.checkAccNum(fromAcc) == False:
        print("invalid from account number")
        fromAcc = int(input(("Enter valid from acc num:")))
    toAcc = int(input("Enter the to acc num: "))
    if toAcc == fromAcc:
        print("Invalid to account number")
        toAcc = int(input(("Enter valid to acc num:")))
    else:
        while self.checkAccNum(toAcc) == False:
            print("invalid to account number")
            toAcc = int(input(("Enter valid to acc num:")))

    amount = input("Enter amount to transfer")
    while self.checkAmount(amount) == False:
        print("invalid transfer amount ")
        amount = input("Enter amount to be deposited:")

    if self.checkAccNum(fromAcc) and self.checkAccNum(toAcc) and self.checkAmount(amount):
        self.transferred += float("{:.2f}".format(int(amount) / 100))
        self.balance -= self.transferred
        print(self.balance)
        print('successfully')
    else:
        self.logout = True
        return self.chooseFunction()
```

In conclusion:

The purpose of this assignment is to build a prototype that has partial functionality, which means it may contain minor mistakes. When coding parts finishes, systematic testing was implemented through testing process. Every possibility test case for each function was spotted and listed, and the team does regression test for each of the function. After testing the correction for major cases, the Output file was generated by the system. Our team has created a tree diagram to represent the structure of our code.

This size of the diagram is considerately large, therefore I have uploaded the original pdf diagram in the zip file