Object Oriented Programming Assignment

C20381946 – Matthew Tweedy

Project Classes & Methods

Customer

```
class Customer(object):
```

```
def __init__(self, custID, name="", age=0) ->
(None):
        self.custID = custID
        self.name = name
        self.age = age
```

This class defines 3 attributes, the customer's unique ID, their name, and their age.

Methods

```
def __str__(self) -> (str):
```

This __str__ method is called when you print the object instance itself e.g. [print(cust1)] . It displays the information I have deemed relevant in a nice format.

```
def addCustomer(self, file=CUSTOMERS):
```

This method reads how many lines are in the file, and sets this number to the current customer's ID (custID) because I have written it so that the ID's are indexed at 0 (the first customer/account will have an ID of 0).

Then, all the customer attributes/information is written to the customers.txt file.

Account

```
class Account(object):
```

```
def __init__(self, accID, custID, accType="",
balance=0, closed=0):
        self.custID = custID
```

```
        self.accID = accID
        self.accType = accType
        self.balance = balance
        self.closed = closed
```

This class defines 5 attributes. They are the account ID, the ID of the customer who owns the account, the type of the account (savings/checkings), the balance of the account, and whether the account is closed.

Methods

```
def __str__(self):
```

This method displays the account id, the account type and the balance in a nice and simple format.

```
def addAccount(self, file=ACCOUNTS):
```

This method is very similar to addCustomer(). It reads in the amount of lines in the accounts.txt file, and sets new account ID (accID) to this number.

Then writes all the information for this account to the accounts.txt file.

```
def deposit(self, amount, transaction):
```

This method deposits the amount specified by the user into the selected account, so long as it is a number and greater than 0.

Then it takes the transaction object that has been passed and writes its information into the accountsTransactions.txt file.

```
def withdraw(self, amount, transaction):
```

This method withdraws the amount specified by the user from the selected account if it is valid and doesn't exceed the credit limit if it is a Checkings account.

If the account is a Savings account, the user can only make one withdrawal or one transfer per "month". I have coded this in a way that a variable gets increased to 1 if a withdrawal or transfer is made. When the user completely logs out of the Customer, this gets reset to 0. So I have implemented the "month" passing by a user logging out and back in again from their Customer and not with actual timestamps.

```
def transfer(self, amount, receive, transaction):
```

This method calls both the withdraw() and deposit() method, as it takes the specified amount from the users selected account and sends it to another chosen account.

```
def displayTransactions(self):
```

This method finds how many lines are in the accountsTransactions.txt file, finds the list of transaction IDs that contain the current "logged in" customer's account ID (accID). Next it grabs the information of each line from the text file and check if either the account ID receiving or sending the money matches the current customer's account ID. If it does, then it calls the __str__ super method for the transaction to display all relevant information about the transaction.

```
def deleteAccount(self):
```

This method updates the "closed" attribute for the account instance from 0 to 1 and then writes the updated line to the accounts.txt file.

This change means that users will not be able to access or send money to the closed account.

```
def showBalance(self):
```

Returns the current balance of the account.

SavingsAccount

```
class SavingsAccount(Account):
```

```
def __init__(self, accID=0, custID=0, accType="savings", balance=0, closed=0):
        super().__init__(accID, custID, accType, balance, closed)
```

This class inherits all methods and attributes from the Customer parent class, however we preset the "accType" attribute to "savings" to distinguish this as a Savings Account.

It contains a protected variable "balance", as we use this to check if the balance after a deposit(), withdraw() or transfer() call would leave balance with an invalid value.

```
def __str__(self):
```

This inherits the same __str__() method from the Customer parent class, just displays that the account is a Savings Account in the title.

CheckingsAccount

```
class CheckingsAccount(Account):
```

```
def __init__(self, accID=0, custID =0,  accType="checkings", balance=0, closed=0):
        super().__init__(accID, custID, accType, balance, closed)
```

Same as SavingsAccount, presets “accType” to “checkings”.

The same as in SavingsAccount, it contains the protected variable “balance”.

Checkings Accounts are allowed to have a negative balance, up to a specified credit limit. I have set this to 300 (or -300 really). If the new balance would go past -300, the method handles this.

```
def __str__(self):
```

This inherits the same __str__() method from the Customer parent class, just displays that the account is a Checkings Account in the title.

Transaction

```
class Transaction(object):
```

```
def __init__(self, transacType = "deposit",
transacID=0, accid=0 , acc_receive_ID=0, amount=0):
        self.id = transacID
        self.transaction_type = transacType
        self.acc_id = accid
        self.amount = amount
        self.rec_acc = acc_receive_ID
```

This class stores information like the transaction type, which account the money is going to, which account the money is coming from and how much money is being sent.

Methods

```
def line_generate(self, id):
```

This method generates and returns the transaction string to be written to the accountsTransactions.txt file.

Exception Classes

```
class BalanceTooLow(Exception):
    """Balance too low in acount"""
class CreditTooLow(Exception):
```

```
    """Credit too low in account"""
class InvalidTransfer(Exception):
    """Transfer was disallowed"""
```

Each of these classes are used to signify a state for the “balance” attribute and can be raised when certain conditions are met, for example the user not trying to withdraw more money from their account than what is in the account.

User Manual Below

User Manual

MAIN MENU:

- Login to Customer
  - You will be prompted to enter an ID from the current list of account IDs. You will be sent to the **LOGGED IN MENU**

- Create Customer
  - You will be prompted to enter name and age. Will log you into your new Customer

- Exit
  - Ends the program and displays a goodbye message

LOGGED IN MENU:

- Create New Account
  - You will be prompted to choose to choose one of the following

- Savings Account
  - Creates a new **Savings** Account for you
- Checkings Account
  - Creates a new **Checkings** Account you
- Go back to Menu
  - Returns you to **LOGGED IN MENU**

❖ Select an Account
- Prints a list of the accounts you currently have. It displays the type of account, the account ID, and the balance of the account. It also indicates if the accounts are open or closed. Prints **SELECTED ACCOUNT MENU**

❖ Logout
- Returns you to **MAIN MENU**

SELECTED ACCOUNT MENU:

❖ Display Account Details
- Displays the account ID, the account type, and the account balance

❖ View Balance
- Displays the current balance for the account

❖ Deposit
- You will be prompted for a numeric value to add to your account balance

❖ Withdraw
- You will be prompted for a numeric value to remove from their account balance
  - If the account you are accessing is a Savings Account, you can only make one withdrawal or transfer per month

❖ Transfer
- Displays the **TRANSFER MENU**
  - If the account you are accessing is a Savings Account, you can only make one withdrawal or transfer per month

❖ View Transactions
- Displays all the transactions recorded for the current logged in account. If there are none, it will tell you

❖ Close Account
- Closes the current account you are accessing and will return you to **LOGGED IN MENU**

- Exit
  - Returns you to **LOGGED IN MENU**

TRANSFER MENU:

- Displays the list of all accounts that are not closed, including their type, their account ID and the name of the Customer who owns the account.
- You will be prompted to enter the ID of the account you wish to transfer money to. You will then be asked to enter the amount of money to transfer. The transaction will then take place.
- You will then be returned to **SELECTED ACCOUNT MENU**

Difficulties

In this project I found a few things difficult to figure out and implement. These include how multiple objects can be used to interact with each other and writing the transactions for transfers. Grasping the concept of laying out my attributes in such a way, that from each class I can get all the necessary information I want, took me quite a while. I had to trial and error many different logical solutions until I figured out what worked with my strings that I was writing to the files. This was connected to my issue with the writing of the transactions for transfers, as I had fully implemented a working version for deposit and withdraw, however once I tried to include transfers transactions, it was duplicating them on, or writing two on the same line. I spent a while debugging until I had a new version that worked, however this was one of the main struggles for me, as there were so many parts of the code where I had to pay attention to what the code was doing and if it was updated to be in sync with my changes I had made in the classes etc.