BANK REPORT

Stephen Hughes - D21126653

TECHNOLOGICAL UNIVERCITY DUBLIN
Grangegorman

Introduction

I decided to choose the banking system as I was most familiar with this project from having attended the lab that was similar. However, there was a difficult task of instead of just keeping details of a user within a list, I now had to save the details for re-use in text files so that a user may log back into their account.

To tackle my project, I first decided to do a simple ToDo list of functions I wanted within my program, I found this rather helpful as any time I got stuck on a function for too long I moved along to another one which let my mind restart from a perspective different to the last. Moving onto a new section often let me kick-start a new problem with prior experience and allowed me to progress than stay stuck on one problem.

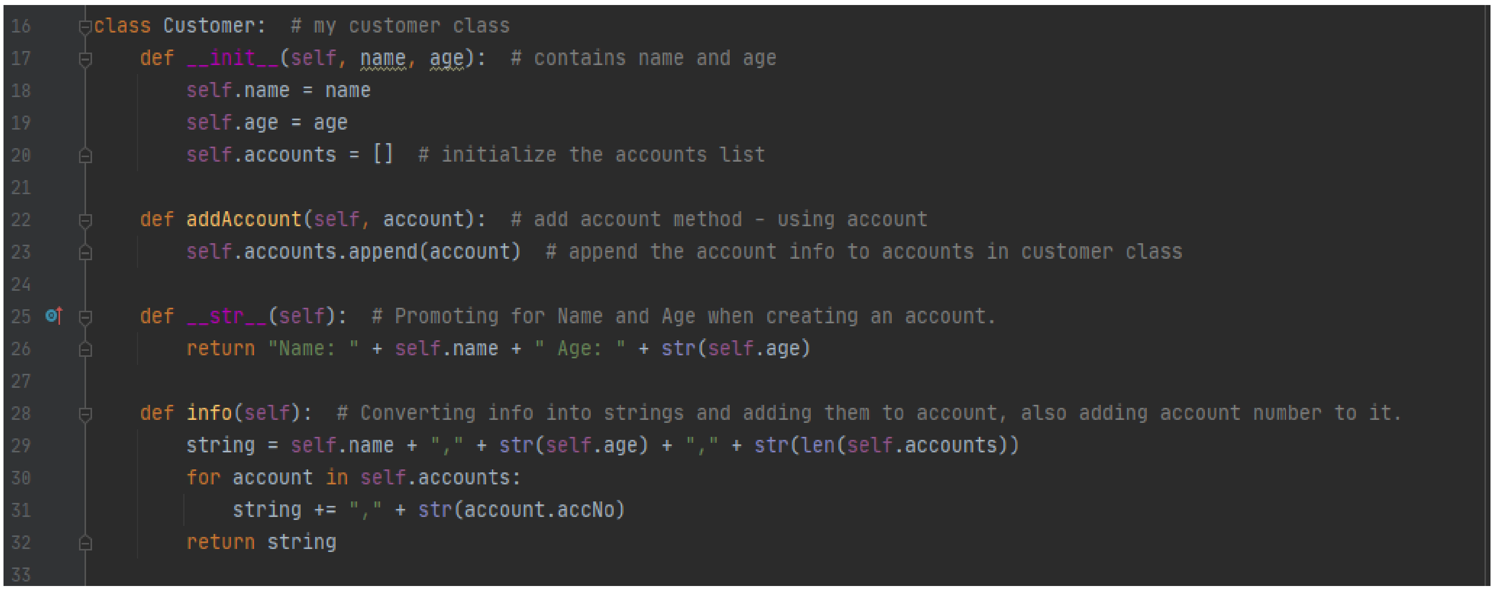
ToDo

```
ToDo:

Customer Class:
Name Age - Verify Age

Accounts Class:
Balance
Account Number
Type
Contains Deposit
Contains Transfer
Contains Withdraw

Checking Account Class:
Splits inherit information.
Contains transfer with limit
Contains Withdraw with limit

Savings Account Class:
Contains withdraw with Type
Contains transfer with Type

Read/Write

Menu with Transaction writes

Exit
```

Sections of Code + Explanation

I started with the Customer class. I found it logical to start with the user who was going to be interacting with the system and then work up the system they plan to interact with bit by bit.

Within my customer would be their name and age, I found this was the information needed to assign to a customer within this class as later-on they will be assigned an account number. The age was used later to also verify if they could open a certain account, such as a savings or checking etc.

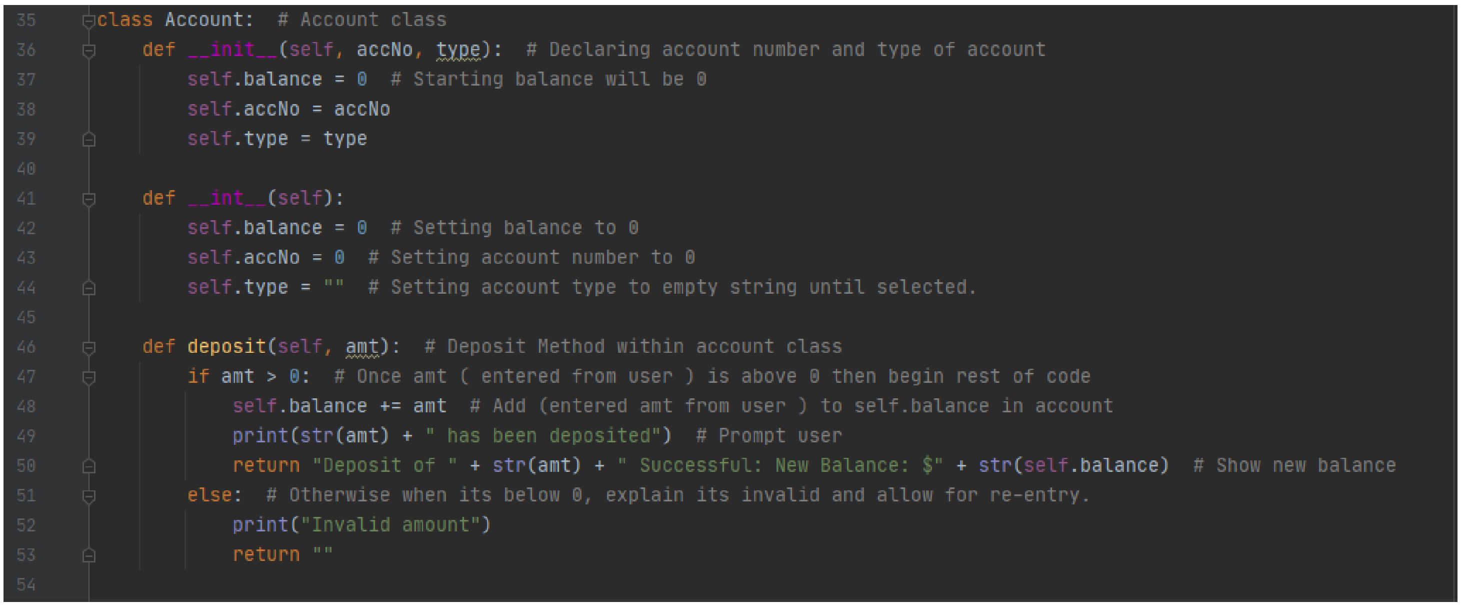


```python
class Customer:  # my customer class
    def __init__(self, name, age):  # contains name and age
        self.name = name
        self.age = age
        self.accounts = []  # initialize the accounts list

    def addAccount(self, account):  # add account method - using account
        self.accounts.append(account)  # append the account info to accounts in customer class

    def __str__(self):  # Promoting for Name and Age when creating an account.
        return "Name: " + self.name + " Age: " + str(self.age)

    def info(self):  # Converting info into strings and adding them to account, also adding account number to it.
        string = self.name + "," + str(self.age) + "," + str(len(self.accounts))
        for account in self.accounts:
            string += "," + str(account.accNo)
        return string

```

Accounts Class

My second class was the Account class as it co-existed with the previous class, taking in the users accountNo. I first declared the accNo and Type then set their default values to empty string or 0.

I then began with the deposit method. (my comments are more neatly displayed in the program I believe word is changing the formatting) My deposit method would first check if the amount entered by the user is greater than 0, and if it is it would allow the rest of the code to run. Upon running it would set the balance within the account class to the amount entered by the user, then prompt them that they've added their cash, it would then return their new balance after entry to clarify to them what their total is.

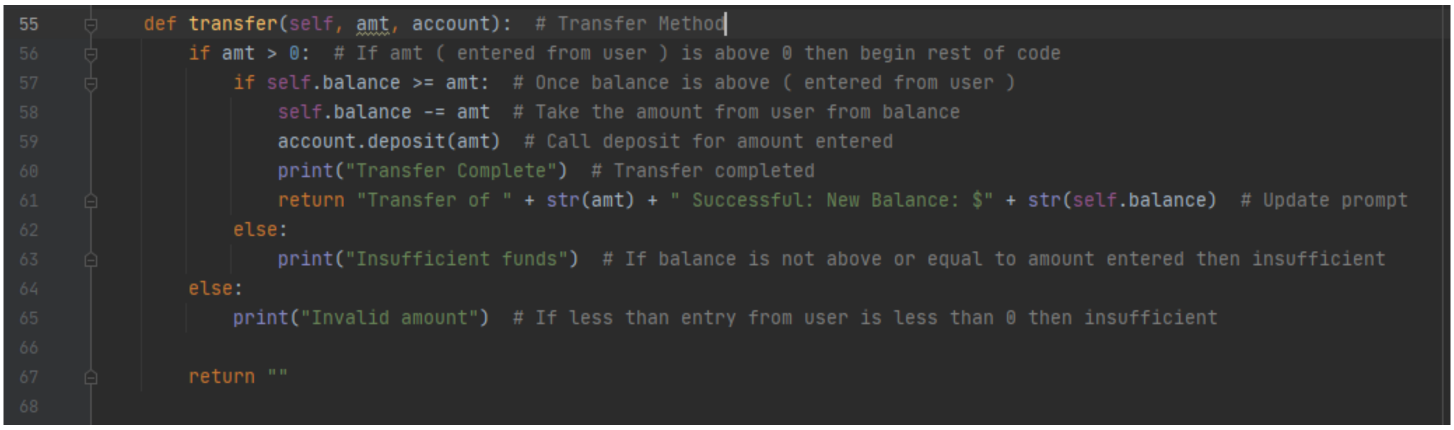


```python
class Account:  # Account class
    def __init__(self, accNo, type):  # Declaring account number and type of account
        self.balance = 0  # Starting balance will be 0
        self.accNo = accNo
        self.type = type

    def __int__(self):
        self.balance = 0  # Setting balance to 0
        self.accNo = 0  # Setting account number to 0
        self.type = ""  # Setting account type to empty string until selected.

    def deposit(self, amt):  # Deposit Method within account class
        if amt > 0:  # Once amt ( entered from user ) is above 0 then begin rest of code
            self.balance += amt  # Add (entered amt from user ) to self.balance in account
            print(str(amt) + " has been deposited")  # Prompt user
            return "Deposit of " + str(amt) + " Successful: New Balance: $" + str(self.balance)  # Show new balance
        else:  # Otherwise when its below 0, explain its invalid and allow for re-entry.
            print("Invalid amount")
            return ""

```

Still within the account class is my transfer method, after the deposit. It is a similar idea to the previous where it will check if the amt (amount entered from user) is greater than 0, then proceed.

It will then take the amount entered from the user's balance, as they are transferring their money. And then call the deposit method for the amount entered. This will be further explained later as when they chose this option is when they select which account to transfer to. The program will then prompt the user with a successful and new balance message.

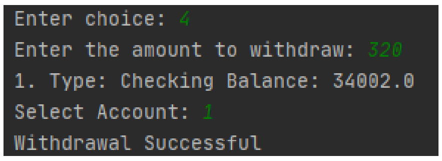
The two if checks are first to check if its above 0, display invalid amount if not and if the balance is higher than the amount the user wishes to transfer, as they would have insufficient funds.



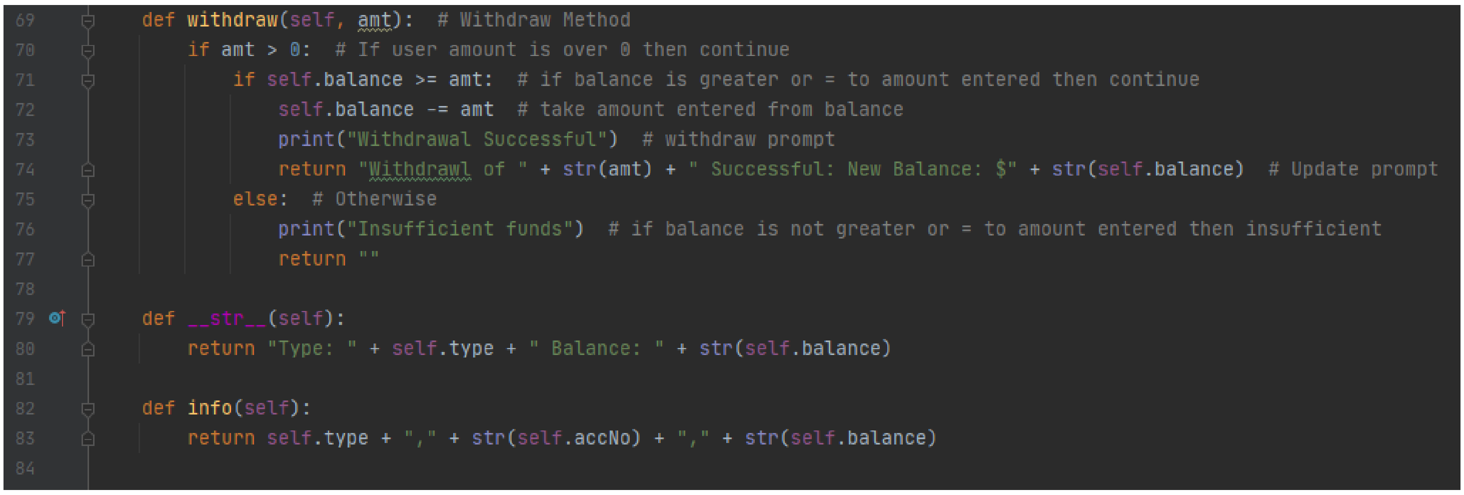
```
    def transfer(self, amt, account):  # Transfer Method
        if amt > 0:  # If amt ( entered from user ) is above 0 then begin rest of code
            if self.balance >= amt:  # Once balance is above ( entered from user )
                self.balance -= amt  # Take the amount from user from balance
                account.deposit(amt)  # Call deposit for amount entered
                print("Transfer Complete")  # Transfer completed
                return "Transfer of " + str(amt) + " Successful: New Balance: $" + str(self.balance)  # Update prompt
            else:
                print("Insufficient funds")  # If balance is not above or equal to amount entered then insufficient
        else:
            print("Invalid amount")  # If less than entry from user is less than 0 then insufficient

        return ""

```

Next within the account class was the withdraw method. Very similar to deposit. Same check with the amount entered and same check if the funds are insufficient. Instead of adding to balance it will take away. Prompt user of success. The last message is a display method when selecting a choice in the program. I will give an example below of how it looks. When you select withdraw, deposit, display transactions etc, it will ask you which account then display the type and its current balance. This is to ensure clarity to a user that has multiple accounts open in one bank.

The display for this can be found here:



```
Enter choice: 4
Enter the amount to withdraw: 320
1. Type: Checking Balance: 34002.0
Select Account: 1
Withdrawal Successful
```



```
    def withdraw(self, amt):  # Withdraw Method
        if amt > 0:  # If user amount is over 0 then continue
            if self.balance >= amt:  # if balance is greater or = to amount entered then continue
                self.balance -= amt  # take amount entered from balance
                print("Withdrawal Successful")  # withdraw prompt
                return "Withdrawl of " + str(amt) + " Successful: New Balance: $" + str(self.balance)  # Update prompt
            else:  # Otherwise
                print("Insufficient funds")  # if balance is not greater or = to amount entered then insufficient
                return ""

    def __str__(self):
        return "Type: " + self.type + " Balance: " + str(self.balance)

    def info(self):
        return self.type + "," + str(self.accNo) + "," + str(self.balance)

```

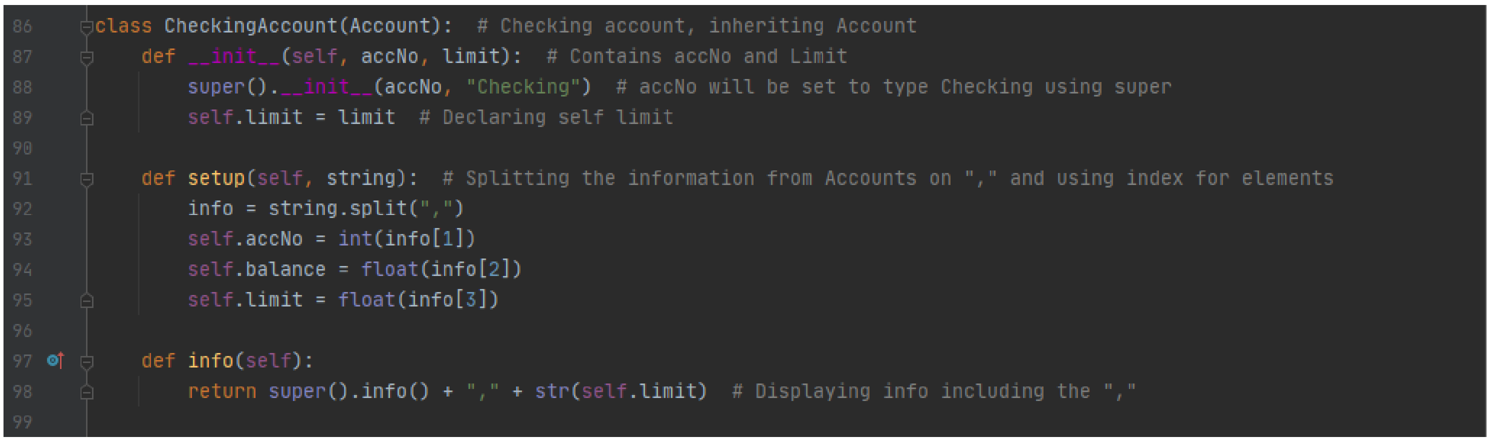
Checking Account Class

Moving onto the Checking Account, this will inherit the Account class from the previous section.

First starting by declaring the limit and account number. Setting the accNo associated when creating the account to a super inherit from Account, removes repetitive code. It will set the accNo associated to be a “Checking” type.

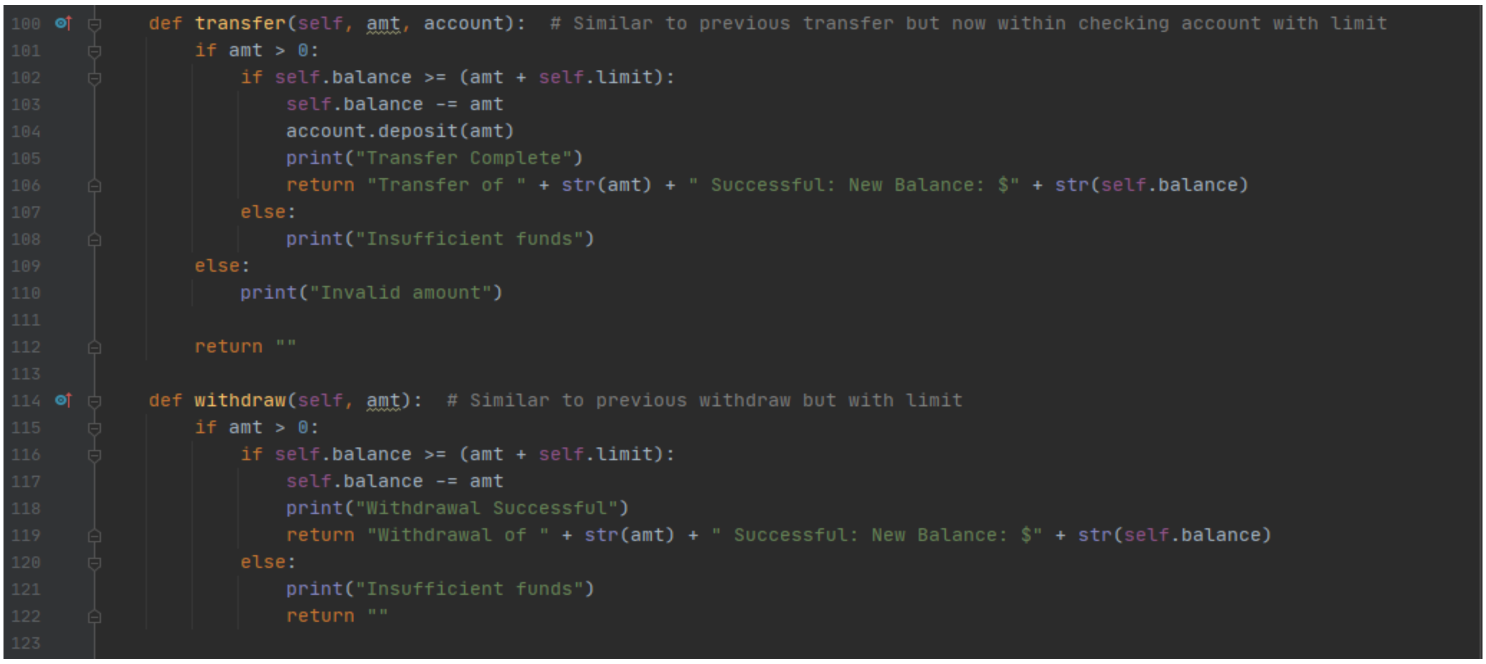
The setup method will separate the information stored in string split on "," and set accNo to be the first index element, balance the second and limit the third index.

The def method, will return the information in info if necessary, including the "," then limit.



```python
class CheckingAccount(Account):  # Checking account, inheriting Account
    def __init__(self, accNo, limit):  # Contains accNo and Limit
        super().__init__(accNo, "Checking")  # accNo will be set to type Checking using super
        self.limit = limit  # Declaring self limit

    def setup(self, string):  # Splitting the information from Accounts on "," and using index for elements
        info = string.split(",")
        self.accNo = int(info[1])
        self.balance = float(info[2])
        self.limit = float(info[3])

    def info(self):
        return super().info() + "," + str(self.limit)  # Displaying info including the ","

```

Then within my Checking Account I have transfer and withdraw methods. These are used instead of the previous as they need to now add the limit to the functions check statement.

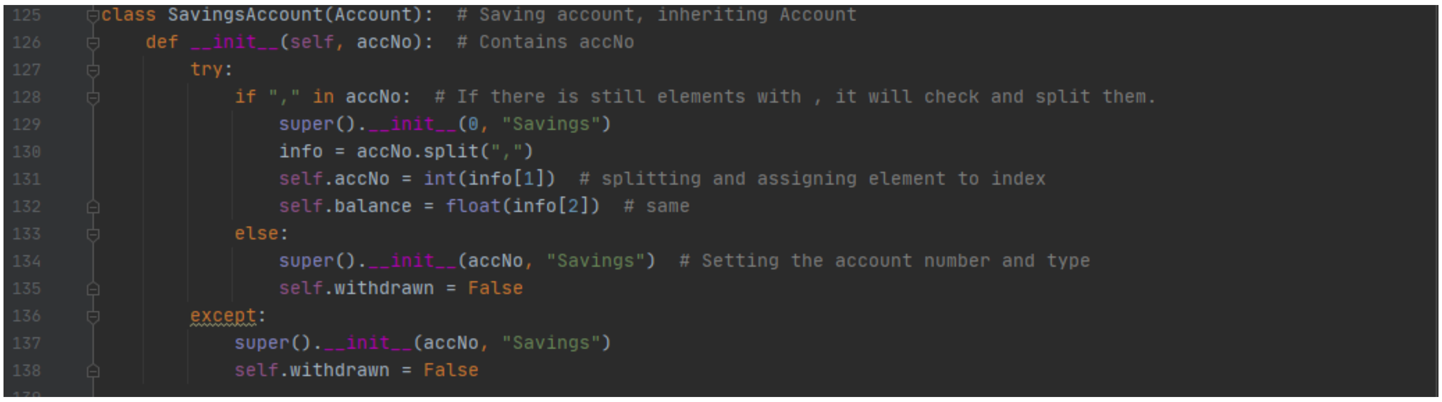


```python
    def transfer(self, amt, account):  # Similar to previous transfer but now within checking account with limit
        if amt > 0:
            if self.balance >= (amt + self.limit):
                self.balance -= amt
                account.deposit(amt)
                print("Transfer Complete")
                return "Transfer of " + str(amt) + " Successful: New Balance: $" + str(self.balance)
            else:
                print("Insufficient funds")
        else:
            print("Invalid amount")

        return ""

    def withdraw(self, amt):  # Similar to previous withdraw but with limit
        if amt > 0:
            if self.balance >= (amt + self.limit):
                self.balance -= amt
                print("Withdrawal Successful")
                return "Withdrawal of " + str(amt) + " Successful: New Balance: $" + str(self.balance)
            else:
                print("Insufficient funds")
                return ""

```

Saving Account Class

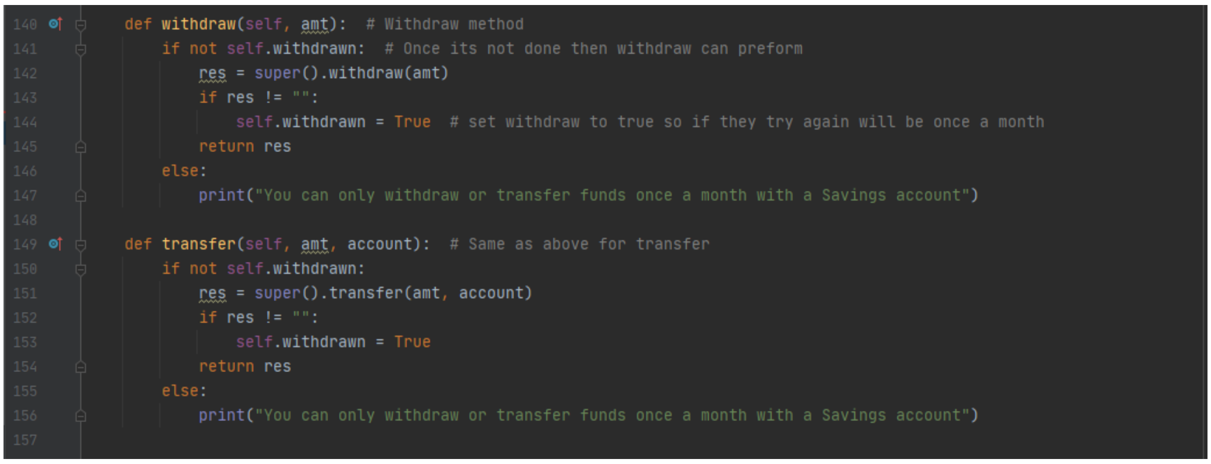
Next is the Saving Account class, this will contain an inherit of Account just like the checking account.

It will also be similar to the checking account, split the information on, and store it into info based on the index element. It first checks if there is a, in accNo, if so, it will store that information and split it. Otherwise, it will just set the account number to the type of savings. With its status of if it has already done its monthly withdraw too false.



```python
class SavingsAccount(Account):  # Saving account, inheriting Account
    def __init__(self, accNo):  # Contains accNo
        try:
            if "," in accNo:  # If there is still elements with , it will check and split them.
                super().__init__(0, "Savings")
                info = accNo.split(",")
                self.accNo = int(info[1])  # splitting and assigning element to index
                self.balance = float(info[2])  # same
            else:
                super().__init__(accNo, "Savings")  # Setting the account number and type
                self.withdrawn = False
        except:
            super().__init__(accNo, "Savings")
            self.withdrawn = False
```

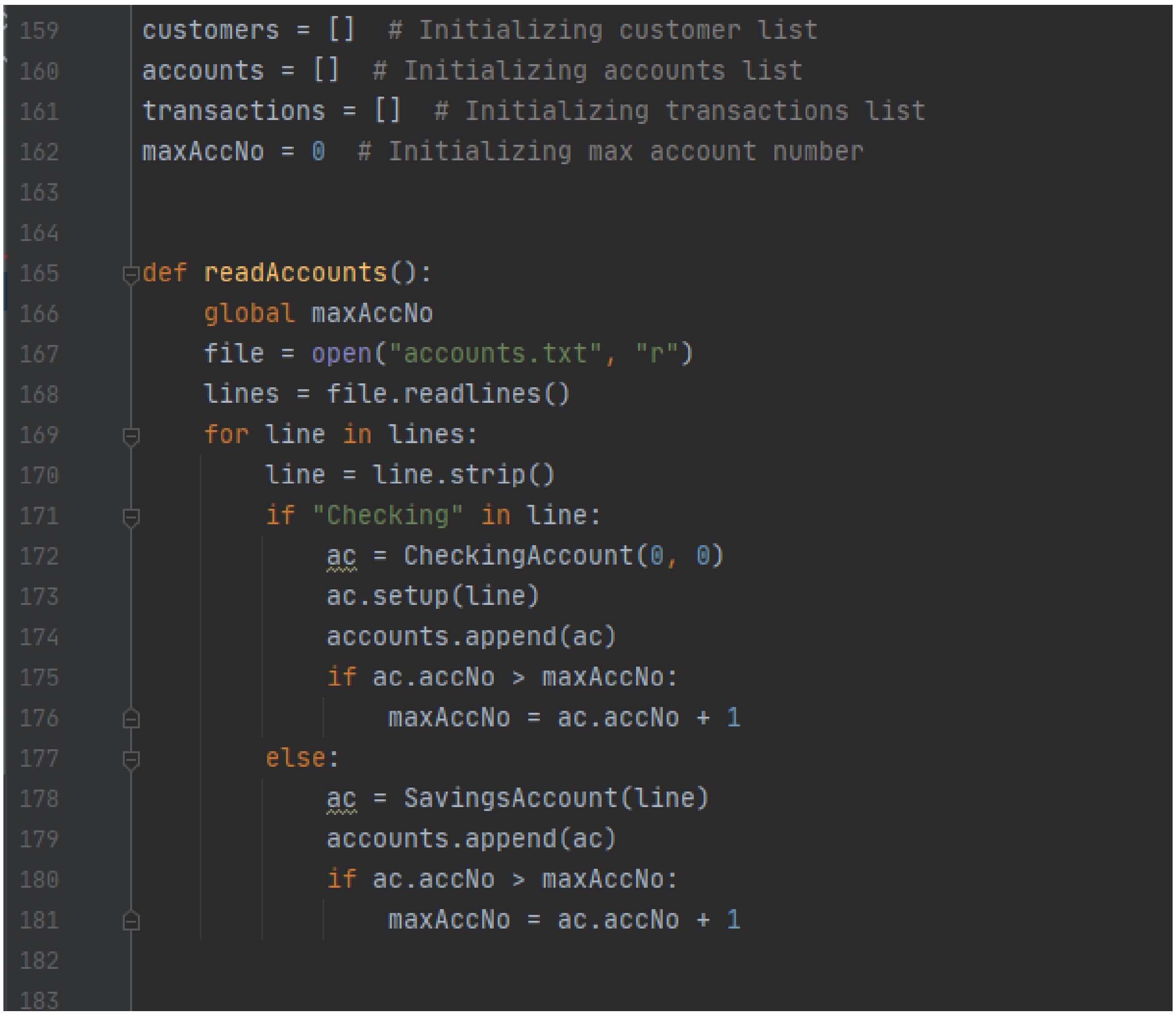
Within the saving account is also the withdraw and transfer methods as they have an additional feature. They will check if the user has already withdrawn for the month. Using supers to get the information from amt and account. Displaying to user they that can only withdraw / transfer once per month into a savings when withdrawn is = True.



```python
    def withdraw(self, amt):  # Withdraw method
        if not self.withdrawn:  # Once its not done then withdraw can preform
            res = super().withdraw(amt)
            if res != "":
                self.withdrawn = True  # set withdraw to true so if they try again will be once a month
            return res
        else:
            print("You can only withdraw or transfer funds once a month with a Savings account")

    def transfer(self, amt, account):  # Same as above for transfer
        if not self.withdrawn:
            res = super().transfer(amt, account)
            if res != "":
                self.withdrawn = True
            return res
        else:
            print("You can only withdraw or transfer funds once a month with a Savings account")
```

Reading and Writing Files

First, I had to add my list initializers outside of the classes of my code. Then I began on the methods to read the information. It will first open the accounts file as a read, it will strip the line and check if Checking has been found inside the file, if it has, it will set the values of the account to 0, 0 then if that number already exists it will set the accNo to be +1, in turn this will create a new customer each time an account is opened for Checking. Otherwise, it will do the same for a saving account but does not need to word search for Savings.

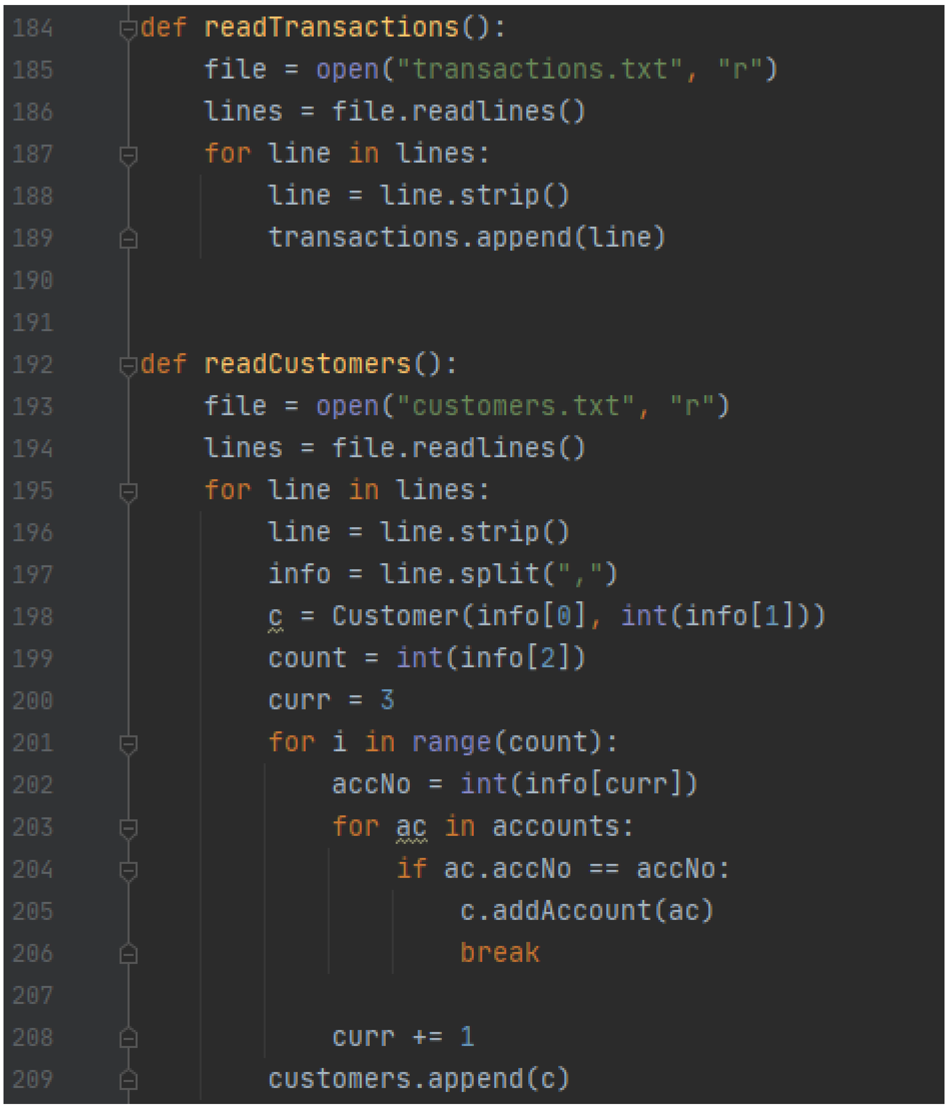


```
customers = []  # Initializing customer list
accounts = []  # Initializing accounts list
transactions = []  # Initializing transactions list
maxAccNo = 0  # Initializing max account number


def readAccounts():
    global maxAccNo
    file = open("accounts.txt", "r")
    lines = file.readlines()
    for line in lines:
        line = line.strip()
        if "Checking" in line:
            ac = CheckingAccount(0, 0)
            ac.setup(line)
            accounts.append(ac)
            if ac.accNo > maxAccNo:
                maxAccNo = ac.accNo + 1
        else:
            ac = SavingsAccount(line)
            accounts.append(ac)
            if ac.accNo > maxAccNo:
                maxAccNo = ac.accNo + 1


```

Next is the transaction read, it will read the file, strip the line, and append the line to transactions.

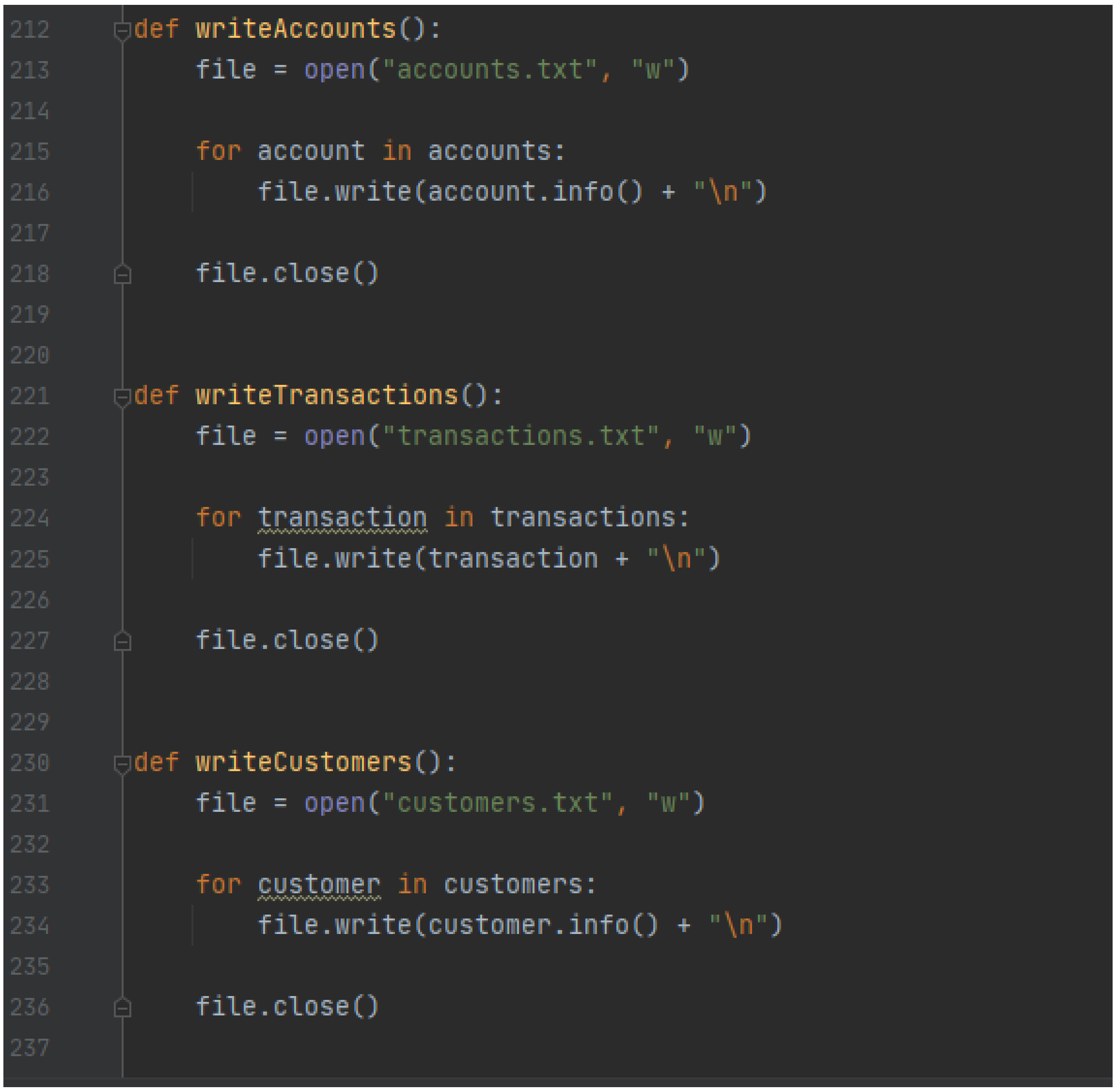
Then the customers, it will set lines to the whole reading of customers and then it will set line to a stripped version of lines, then for if line is in lines, it will split the info from customer on “,” then c will contain the customers info on index 0, 1. With the count being the index of 2.

It will then for I in range of the count from index 2 inside info will create the account if account numbers are the same.

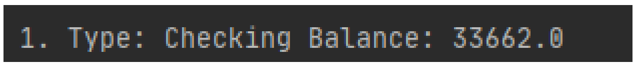


```python
def readTransactions():
    file = open("transactions.txt", "r")
    lines = file.readlines()
    for line in lines:
        line = line.strip()
        transactions.append(line)


def readCustomers():
    file = open("customers.txt", "r")
    lines = file.readlines()
    for line in lines:
        line = line.strip()
        info = line.split(",")
        c = Customer(info[0], int(info[1]))
        count = int(info[2])
        curr = 3
        for i in range(count):
            accNo = int(info[curr])
            for ac in accounts:
                if ac.accNo == accNo:
                    c.addAccount(ac)
                    break

            curr += 1
        customers.append(c)
```

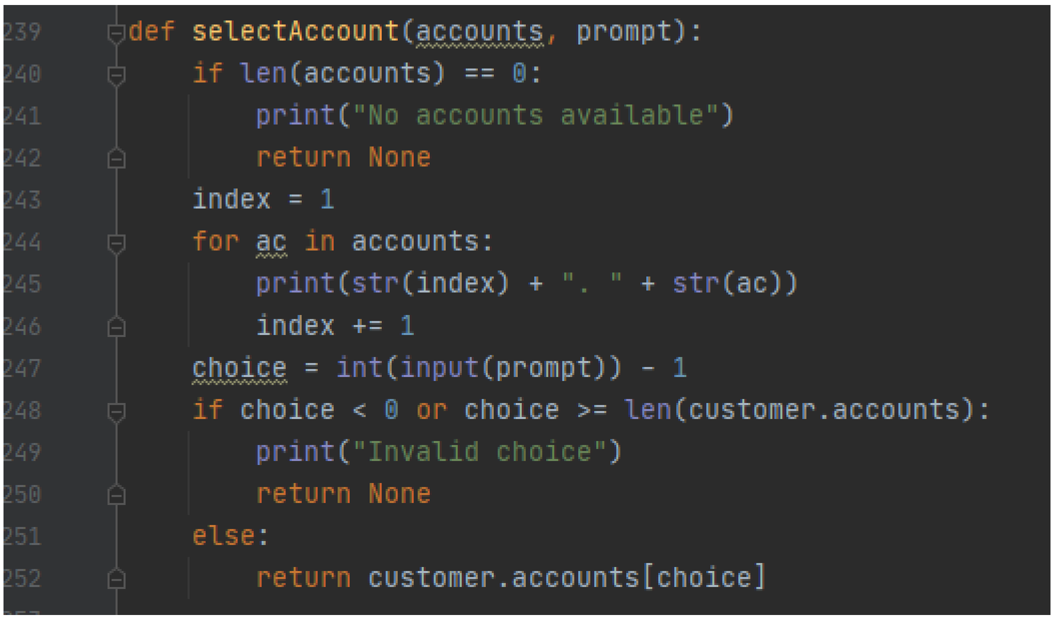
Basic file writing for accounts, transaction, and customer. It will file write then leave a new line for the next account to be made, it makes it easier to read. Will also close the file.



```python
def writeAccounts():
    file = open("accounts.txt", "w")

    for account in accounts:
        file.write(account.info() + "\n")

    file.close()


def writeTransactions():
    file = open("transactions.txt", "w")

    for transaction in transactions:
        file.write(transaction + "\n")

    file.close()


def writeCustomers():
    file = open("customers.txt", "w")

    for customer in customers:
        file.write(customer.info() + "\n")

    file.close()

```

Method to select an account, when selecting an account, it will check if the length of accounts is 0, if it is then there are none available. Otherwise, it will print the index of the account that the user may select. Like so: 

```
1. Type: Checking Balance: 33662.0
```

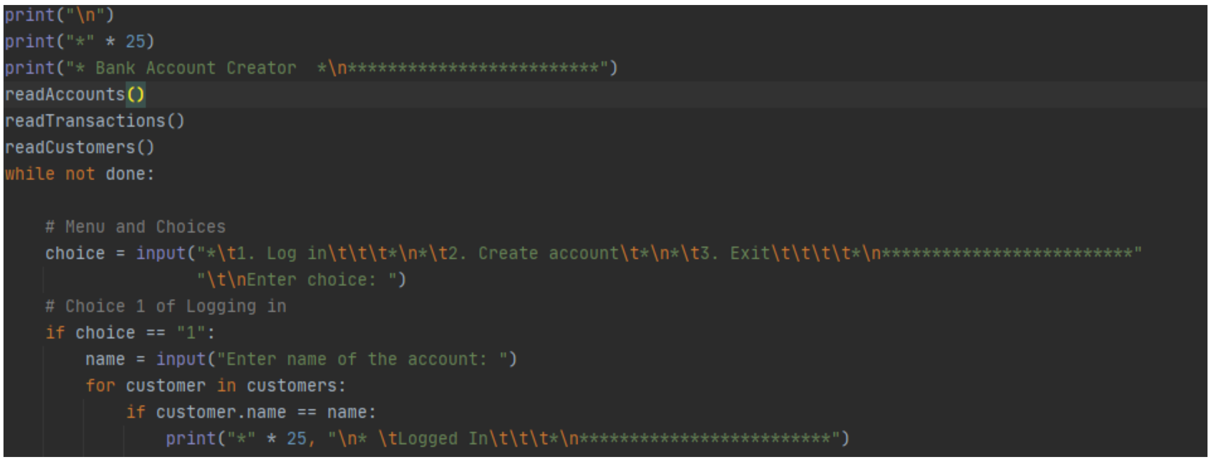


```python
def selectAccount(accounts, prompt):
    if len(accounts) == 0:
        print("No accounts available")
        return None
    index = 1
    for ac in accounts:
        print(str(index) + ". " + str(ac))
        index += 1
    choice = int(input(prompt)) - 1
    if choice < 0 or choice >= len(customer.accounts):
        print("Invalid choice")
        return None
    else:
        return customer.accounts[choice]
```

Menu prompt

Now begins the prompt / user interaction functionality. It starts with a square menu to show that it's a Bank Account system. I also have it reading the files directly after this, as any information will then be already saved and be ready to be used. It then prompts the user to either log in, create an account or exit. When choice is 1 it will log in when 2 it will create etc.

Will prompt user for name of the account to log-in and display logged in if the name is correct.

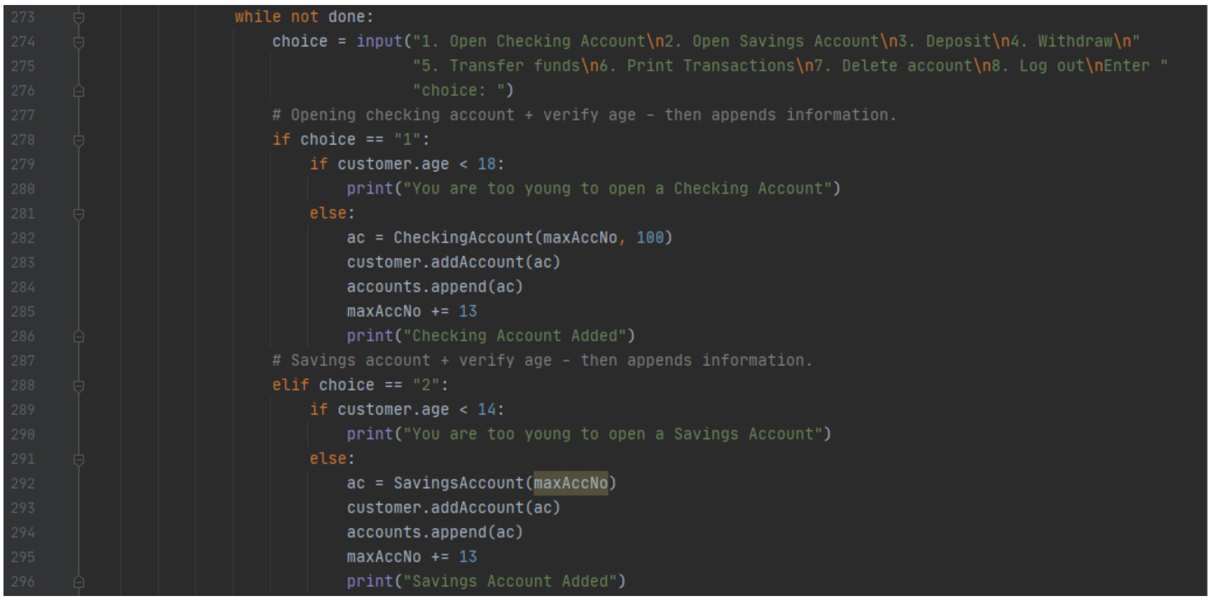


```python
print("\n")
print("*" * 25)
print("* Bank Account Creator  *\n*************************")
readAccounts()
readTransactions()
readCustomers()
while not done:

    # Menu and Choices
    choice = input("*\t1. Log in\t\t\t*\n*\t2. Create account\t*\n*\t3. Exit\t\t\t\t*\n*************************"
                   "\t\nEnter choice: ")
    # Choice 1 of Logging in
    if choice == "1":
        name = input("Enter name of the account: ")
        for customer in customers:
            if customer.name == name:
                print("*" * 25, "\n* \tLogged In\t\t\t*\n*************************")
```

After logging in, it will give the options to open accounts, deposit, withdraw, transfer, print, delete or log out.

When they chose option 1 it will verify the age, they entered is above 18. If not will say they are too young. Otherwise continue, it will then set the unique values of each account and create the account.

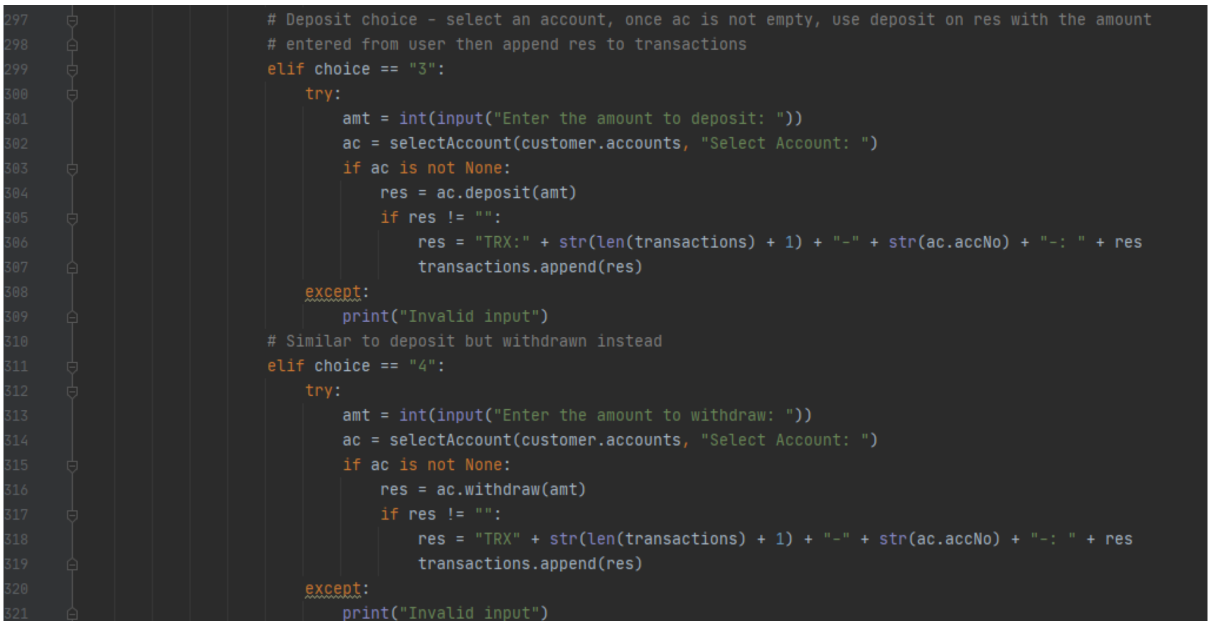
Same logic for savings account, by verifying age then adding unique values to the account created.



```python
            while not done:
                choice = input("1. Open Checking Account\n2. Open Savings Account\n3. Deposit\n4. Withdraw\n"
                               "5. Transfer funds\n6. Print Transactions\n7. Delete account\n8. Log out\nEnter "
                               "choice: ")
                # Opening checking account + verify age - then appends information.
                if choice == "1":
                    if customer.age < 18:
                        print("You are too young to open a Checking Account")
                    else:
                        ac = CheckingAccount(maxAccNo, 100)
                        customer.addAccount(ac)
                        accounts.append(ac)
                        maxAccNo += 13
                        print("Checking Account Added")
                # Savings account + verify age - then appends information.
                elif choice == "2":
                    if customer.age < 14:
                        print("You are too young to open a Savings Account")
                    else:
                        ac = SavingsAccount(maxAccNo)
                        customer.addAccount(ac)
                        accounts.append(ac)
                        maxAccNo += 13
                        print("Savings Account Added")
```

Choice 3 is to deposit; it will ask them to select the account to deposit to. It will also give display for each transaction at the end of each withdraw, saving it under TRX and then the value of the transaction length + 1 each time. With the account number representing it also.

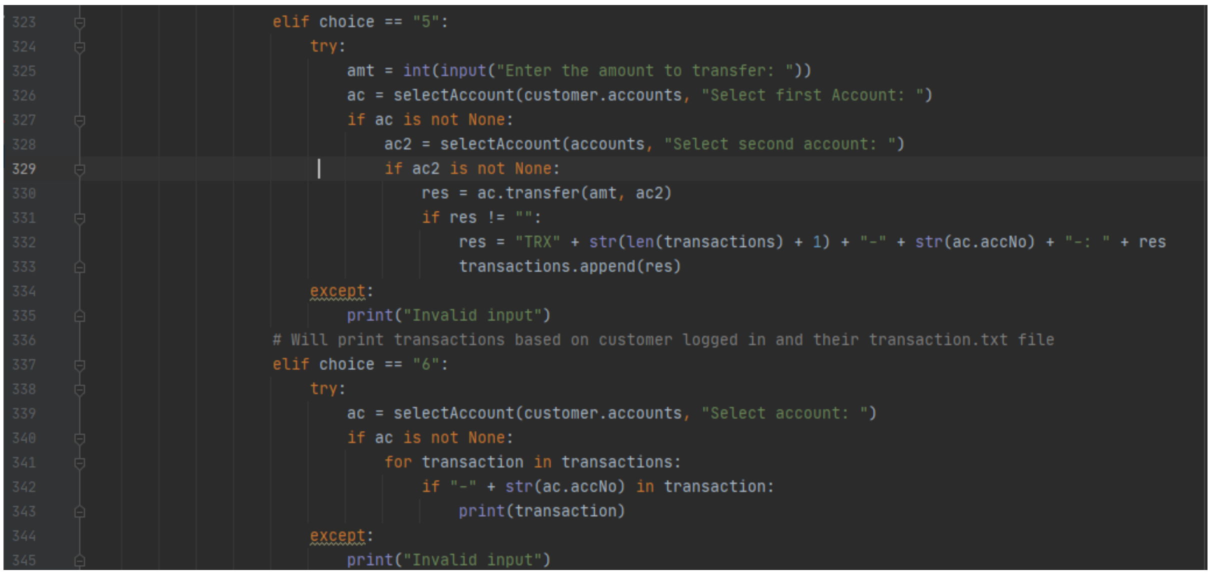
The same as above is done but for withdraw, ask the user the amount then ask the user to select accounts then save the transaction done.



```python
                # Deposit choice - select an account, once ac is not empty, use deposit on res with the amount
                # entered from user then append res to transactions
                elif choice == "3":
                    try:
                        amt = int(input("Enter the amount to deposit: "))
                        ac = selectAccount(customer.accounts, "Select Account: ")
                        if ac is not None:
                            res = ac.deposit(amt)
                            if res != "":
                                res = "TRX:" + str(len(transactions) + 1) + "-" + str(ac.accNo) + "-: " + res
                                transactions.append(res)
                    except:
                        print("Invalid input")
                # Similar to deposit but withdrawn instead
                elif choice == "4":
                    try:
                        amt = int(input("Enter the amount to withdraw: "))
                        ac = selectAccount(customer.accounts, "Select Account: ")
                        if ac is not None:
                            res = ac.withdraw(amt)
                            if res != "":
                                res = "TRX" + str(len(transactions) + 1) + "-" + str(ac.accNo) + "-: " + res
                                transactions.append(res)
                    except:
                        print("Invalid input")
```

Choice 5 consists of the transfer, which will ask the amount to transfer, then ask for the second account if it's not None, upon selecting the second account it sends the amount and the ac2 information to transfer method. Then saves this transaction at the end.

Choice 6 will display the accounts transactions that were previously saved to transaction.txt.

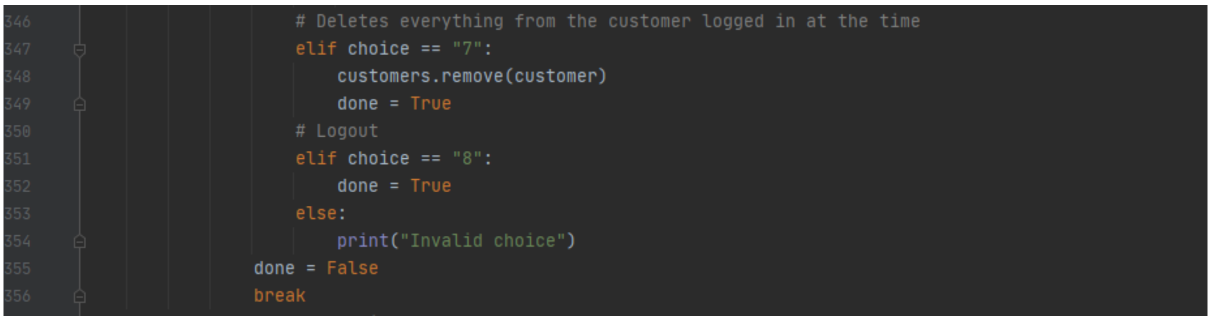


```python
elif choice == "5":
    try:
        amt = int(input("Enter the amount to transfer: "))
        ac = selectAccount(customer.accounts, "Select first Account: ")
        if ac is not None:
            ac2 = selectAccount(accounts, "Select second account: ")
            if ac2 is not None:
                res = ac.transfer(amt, ac2)
                if res != "":
                    res = "TRX" + str(len(transactions) + 1) + "-" + str(ac.accNo) + "-: " + res
                    transactions.append(res)
    except:
        print("Invalid input")
# Will print transactions based on customer logged in and their transaction.txt file
elif choice == "6":
    try:
        ac = selectAccount(customer.accounts, "Select account: ")
        if ac is not None:
            for transaction in transactions:
                if "-" + str(ac.accNo) in transaction:
                    print(transaction)
    except:
        print("Invalid input")
```

Choice 7 will delete the customer list. Setting done to True. Removing them from the system.

Logout will do the same, setting done to true just logging them out.

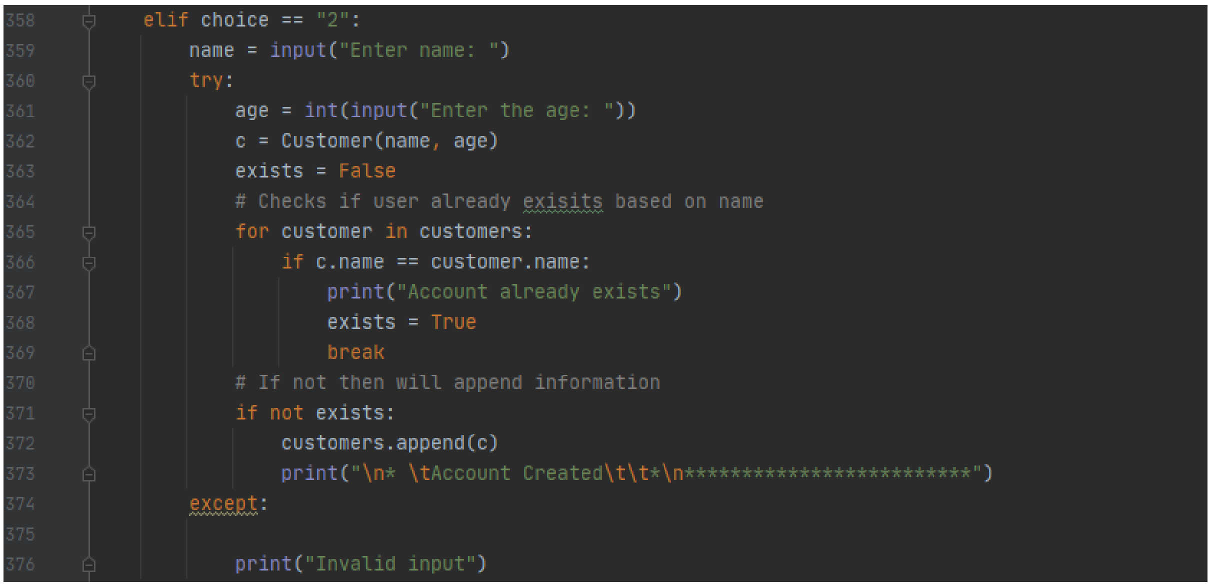
Anything else is invalid.



```python
    # Deletes everything from the customer logged in at the time
    elif choice == "7":
        customers.remove(customer)
        done = True
    # Logout
    elif choice == "8":
        done = True
    else:
        print("Invalid choice")
done = False
break
```

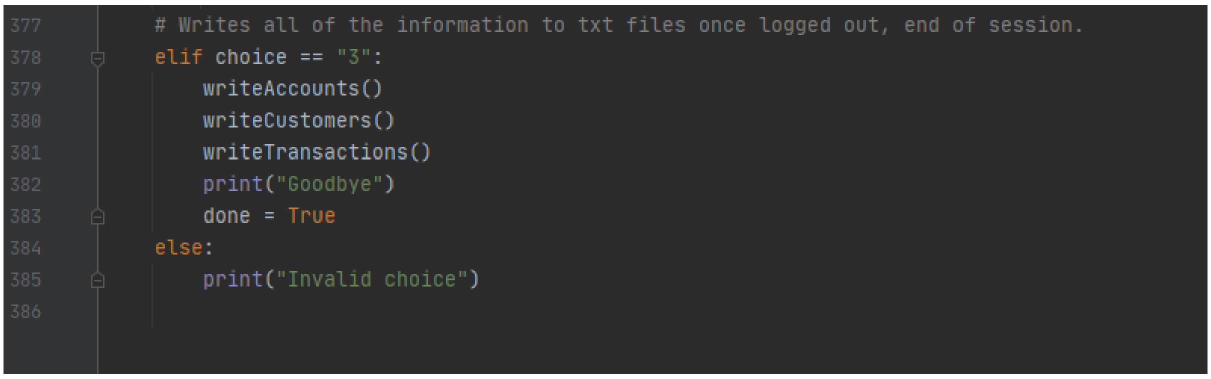
Back to the second choice when given log-in, create or exit. Is choice 2. This is the information that is stored for customer. It will ask for age and name if the name already exists so does the account. Otherwise, it will create the account by appending to customer!

Anything else is invalid.



```
elif choice == "2":
    name = input("Enter name: ")
    try:
        age = int(input("Enter the age: "))
        c = Customer(name, age)
        exists = False
        # Checks if user already exisits based on name
        for customer in customers:
            if c.name == customer.name:
                print("Account already exists")
                exists = True
                break
        # If not then will append information
        if not exists:
            customers.append(c)
            print("\n* \tAccount Created\t\t*\n*************************")
    except:

        print("Invalid input")
```

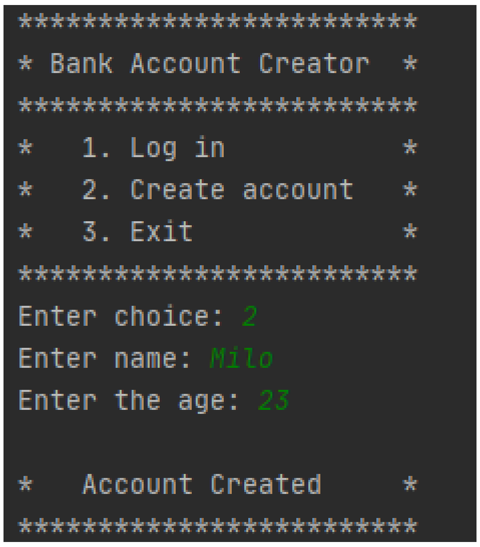
Writes all information once a user logs out completely. This was the best way to get my code to properly work each time a user logged in and tried storing new information.



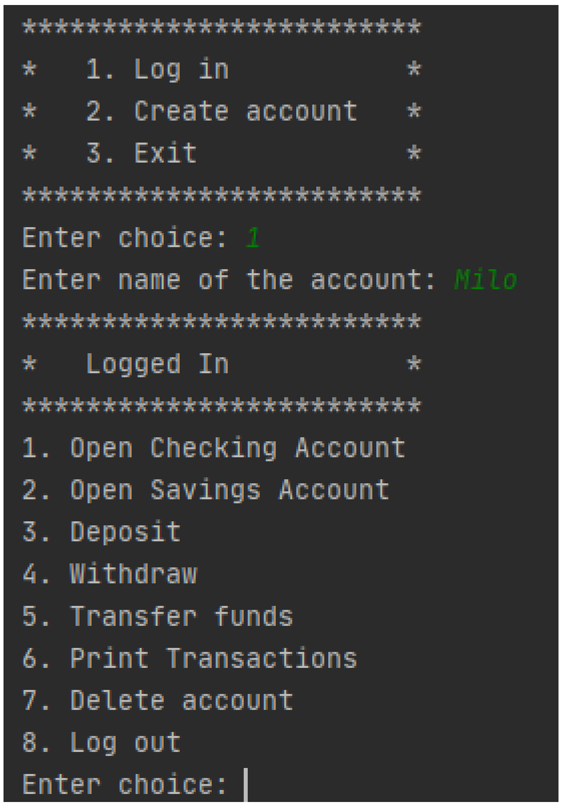
```
# Writes all of the information to txt files once logged out, end of session.
elif choice == "3":
    writeAccounts()
    writeCustomers()
    writeTransactions()
    print("Goodbye")
    done = True
else:
    print("Invalid choice")

```

Displays

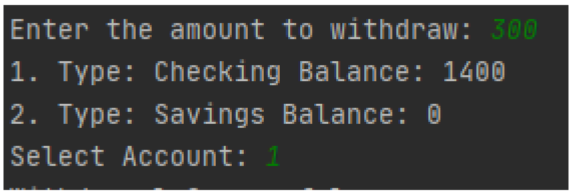
Screenshots of each option of the code.



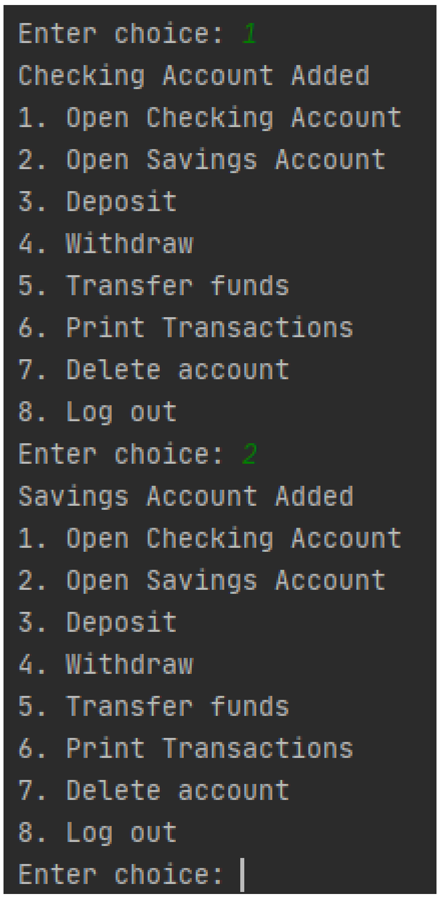
```
*************************
* Bank Account Creator  *
*************************
*   1. Log in           *
*   2. Create account   *
*   3. Exit             *
*************************
Enter choice: 2
Enter name: Milo
Enter the age: 23

*   Account Created     *
*************************
```



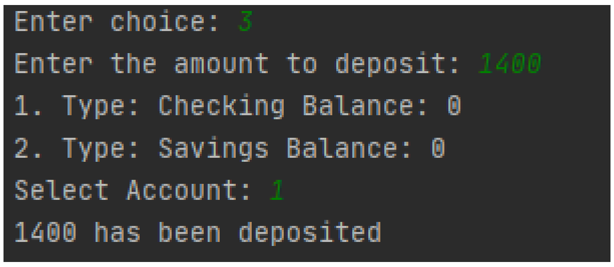
```
*************************
*   1. Log in           *
*   2. Create account   *
*   3. Exit             *
*************************
Enter choice: 1
Enter name of the account: Milo
*************************
*   Logged In           *
*************************
1. Open Checking Account
2. Open Savings Account
3. Deposit
4. Withdraw
5. Transfer funds
6. Print Transactions
7. Delete account
8. Log out
Enter choice: 
```



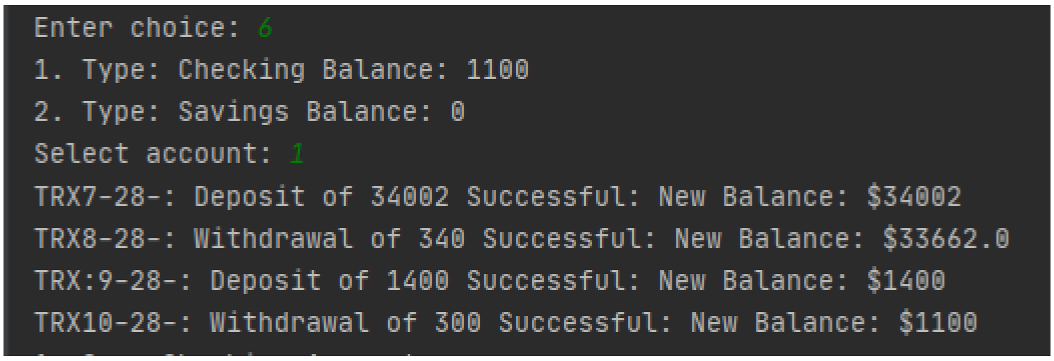
```
Enter the amount to withdraw: 300
1. Type: Checking Balance: 1400
2. Type: Savings Balance: 0
Select Account: 1
```



```
Enter choice: 1
Checking Account Added
1. Open Checking Account
2. Open Savings Account
3. Deposit
4. Withdraw
5. Transfer funds
6. Print Transactions
7. Delete account
8. Log out
Enter choice: 2
Savings Account Added
1. Open Checking Account
2. Open Savings Account
3. Deposit
4. Withdraw
5. Transfer funds
6. Print Transactions
7. Delete account
8. Log out
Enter choice:
```



```
Enter choice: 3
Enter the amount to deposit: 1400
1. Type: Checking Balance: 0
2. Type: Savings Balance: 0
Select Account: 1
1400 has been deposited
```



```
Enter choice: 6
1. Type: Checking Balance: 1100
2. Type: Savings Balance: 0
Select account: 1
TRX7-28-: Deposit of 34002 Successful: New Balance: $34002
TRX8-28-: Withdrawal of 340 Successful: New Balance: $33662.0
TRX:9-28-: Deposit of 1400 Successful: New Balance: $1400
TRX10-28-: Withdrawal of 300 Successful: New Balance: $1100
```