**CISC 327 Assignment 2**

**Breaking Bank**

**Scott Wallace 10051890**

**Brad Guner 10059112**

**Design Document**

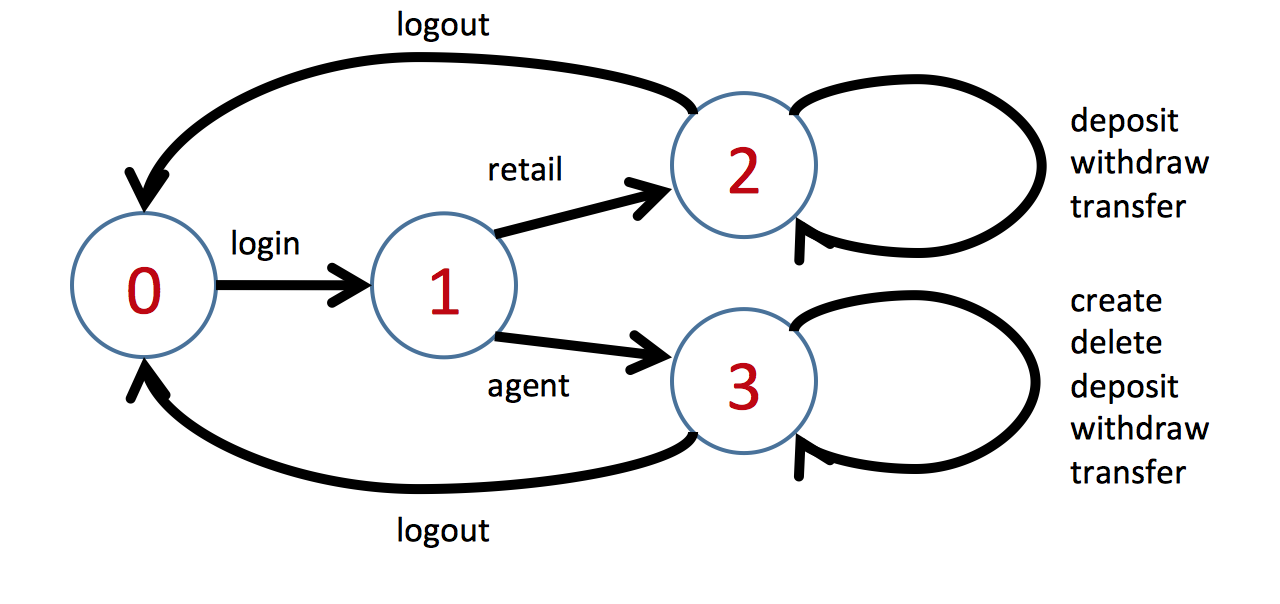
For our architecture we have chosen to go with a Main program, which contains one function, and with 2 classes that handle agent and retail. When our program first runs the main program opens our current accounts file (to be accessed during transactions), and then executes our openBankingSystem function, this function is designed to handle the system login, which type of day to run, and restarting the system after logout. However the function itself does not take the logout command as input, rather receives a logout signal from either a retail or agent day, and then proceeds to restart the system at the pre-login stage.

Our Agent and Retail classes are quite similar in construction and how they operate. The openBankingSystem function instantiates one of these classes and then runs either the runAgentDay or runRetailDay method within the class based on which type was instantiated. These methods are meant to be running to receive any sort of transaction that might occur during a banking day. This is also where the logout command will be received and then returns a logout signal back to openBankingSystem for it to reboot the system.

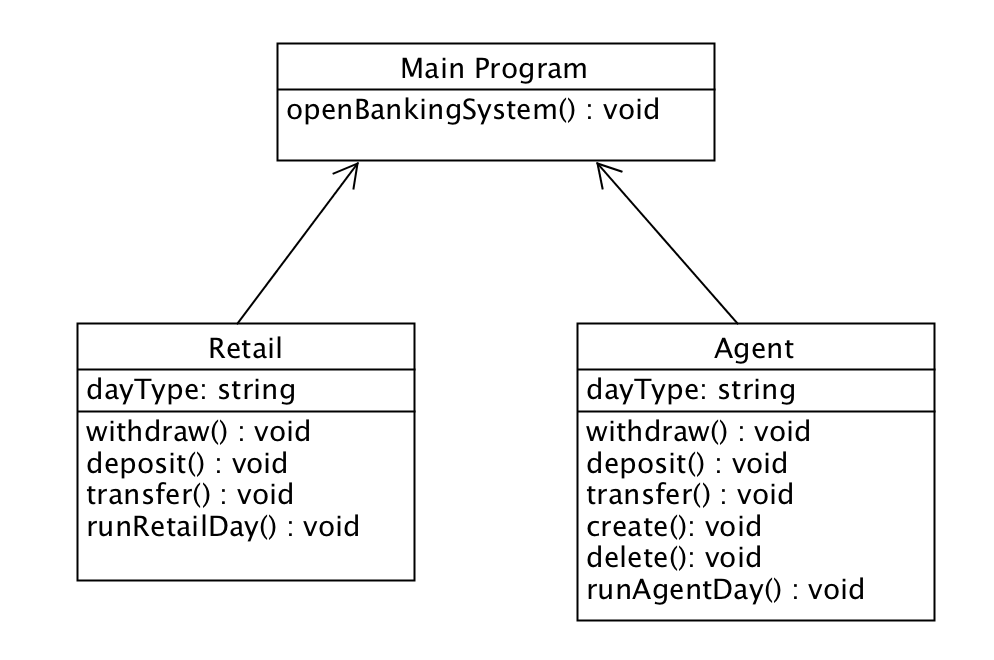
This is architecture is modeled after the 0 to 3 stage architecture (See Figure 1) modeled during class. We decided to implement a similar design because it would make the simplest solution. The 0 and 1 phase of the design are handled by the openBankingSystem function in our main program. Our Retail and Agent classes represent the 2 and 3 phases respectively. The logout functionality is handled in a similar way with either stage 2 or 3 receiving the logout command and starting over in stage 0.

Figure 2 is a simple UML diagram of the programs structure. We’ve taken some liberties in UML standards to illustrate the structure. Main program isn’t actually a class, because of the way python operates. We have chosen to represent it this way to show that openBankingSystem is apart of the main program. Agent and Retail are classes as shown with all the methods they have. The arrows pointing from these classes to main program are not hierarchy but rather symbolize the relationship between main, agent and retail. Since openBankingSystem instantiates one of these classes, we felt it appropriate to represent it in this way.

**Figure 1**

****

**Figure 2**

****

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Function** |
| Main Program | openBankingSystem | Handles login, agent/retail instances (Stage 0 and 1, from Figure 1) |
| Retail | withdraw | Accepts input for a withdraw transaction during Retail, tests for valid input, performs transaction, returns transaction string for summary file. |
| Retail | deposit | Accepts input for a deposit transaction during Retail, tests for valid input, performs transaction, returns transaction string for summary file. |
| Retail | transfer | Accepts input for a transfer transaction during Retail, tests for valid input, performs transaction, returns transaction string for summary file. |
| Retail | runRetailDay | Handles any and all transactions received during a retail day. Tests input and starts the transaction’s method to perform the transaction. Logout is received here and sends logout signal to openBankingSystem.  Our transactions summary file is created and written during this function. |
| Agent | withdraw | Accepts input for a withdraw transaction during Agent, tests for valid input, performs transaction, returns transaction string for summary file. |
| Agent | deposit | Accepts input for a deposit transaction during Agent, tests for valid input, performs transaction, returns transaction string for summary file. |
| Agent | transfer | Accepts input for a transfer transaction during Agent, tests for valid input, performs transaction, returns transaction string for summary file. |
| Agent | create | Accepts input to create account during agent day. Tests that it is able to create the new account. Creates account and returns transaction string for summary file. |
| Agent | delete | Accepts input to delete account during agent day. Tests that it is able to delete the account. Deletes account and returns transaction string for summary file. |
| Agent | runAgentDay | Handles any and all transactions received during a agent day. Tests input and starts the transaction’s method to perform the transaction. Logout is received here and sends logout signal to openBankingSystem.  Our transactions summary file is created and written during this function. |

"""

CISC 327

Breaking Bank

Assignment #2

Scott Wallace 10051890

Brad Guner 10059112

"""

import datetime

import time

import os.path

############################################ RETAIL #################################################

class retail(object):

def \_\_init\_\_(self, type,dailylimit):

self.type = type

self.dailylimit = dailylimit

def withdraw(self):

accNumInput = True

while (accNumInput):

accNum = raw\_input('Account Number: ')

#CHECK TO SEE IF VALID ACCOUNT NUMBER

if (1 == 1): #if account num is valid

amt = True

accNumInput = False

while (amt):

amount = int(input('Withdrawal Amount: '))

amount = amount\*100

if (amount > 100000):

print "Please enter a valid amount."

elif (amount < 0):

print "Please enter a valid amount."

elif (self.dailylimit + amount > 100000):

print "This amount exceeds your daily limit."

else:

self.dailylimit += amount

amt = False

#CREATE STRING TO WRITE TO FILE

accNum = str(accNum)

amount = str(amount)

transactionInfo = '02\_' + accNum + '\_' + amount #NEEDS PROPER FORMATTING STILL

else:

print "Please enter a valid account number."

return transactionInfo

def deposit(self):

accNumInput = True

while (accNumInput):

accNum = raw\_input('Account Number: ')

#CHECK TO SEE IF VALID ACCOUNT NUMBER

if (1 == 1): #if account num is valid

amt = True

accNumInput = False

while (amt):

amount = int(input('Deposit Amount: '))

amount = amount\*100

if (amount > 100000):

print "Please enter a valid amount."

elif (amount < 0):

print "Please enter a valid amount."

else:

amt = False

#CREATE STRING TO WRITE TO FILE

accNum = str(accNum)

amount = str(amount)

transactionInfo = '01\_' + accNum + '\_' + amount #NEEDS PROPER FORMATTING STILL

else:

print "Please enter a valid account number."

return transactionInfo

def transfer(self):

accNumInput = True

accNumInput2 = True

while(accNumInput):

accNumTo = raw\_input('To Account Number: ')

#CHECK to SEE IF FIRST ACCOUNT NUMBER IS VALID

if (1 == 1):

while (accNumInput2):

accNumFrom = raw\_input('From Account Number: ')

#CHECK TO SEE IF SECOND ACCOUNT NUMBER IS VALID

if (1 == 1):

accNumInput = False

accNumInput2 = False

amt = True

while (amt):

amount = int(input('Transfer Amount: '))

amount = amount\*100

if (amount > 100000):

print "Please enter a valid transfer amount."

elif (amount < 0):

print "Please enter a valid transfer amount."

else:

amt = False

#create string for write file

accNumTo = str(accNumTo)

accNumFrom = str(accNumFrom)

amount = str(amount)

transactionInfo = '03\_' + accNumTo + '\_' + accNumFrom + '\_' + amount

else:

print "Please enter a valid account number."

else:

print "Please enter a valid account number."

return transactionInfo

#METHOD WHICH RUNS ANY TRANSACTIONS FOR A RETAIL DAY

#WILL WRITE ANY TRANSACTIONS TO FILE

#LOGOUT IS ACCEPTED AT THIS STAGE

def runRetailDay(self):

running = True

#CREATES TRANSACTION SUMMARY FILE

ts = time.time()

st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

save\_path = './TransactionSummaryFiles/'

file = 'Transaction\_Summary\_File\_\_' + st + '.txt'

filename = file.replace(":", "\_")

completeName = os.path.join(save\_path, filename)

f = open(completeName,'w')

while (running):

#STARTS ACCEPTING RETAIL TRANSACTIONS

transaction = raw\_input('Perform a transaction: ')

transaction.lower()

#TESTS INPUT FOR WHICH TRANSACTION TYPE TO PERFORM

if (transaction == "withdraw"):

newTrans = self.withdraw()

f.write(newTrans + '\n')

elif (transaction == "deposit"):

newTrans = self.deposit()

f.write(newTrans + '\n')

elif (transaction == "transfer"):

newTrans = self.transfer()

f.write(newTrans + '\n')

elif (transaction == "logout"):

f.close()

running = False

else:

print "Please enter a valid transaction type."

return False

###########################################################################################################

############################################ AGENT #################################################

class agent(object):

def \_\_init\_\_(self, type):

self.type = type

def withdraw(self):

accNumInput = True

while (accNumInput):

accNum = raw\_input('Account Number: ')

#CHECK TO SEE IF VALID ACCOUNT NUMBER

if (1 == 1): #if account num is valid

amt = True

accNumInput = False

while (amt):

amount = int(input('Withdrawal Amount: '))

amount = amount\*100

if (amount > 999999):

print "Please enter a valid amount."

elif (amount < 0):

print "Please enter a valid amount."

else:

amt = False

#CREATE STRING TO WRITE TO FILE

accNum = str(accNum)

amount = str(amount)

transactionInfo = '02\_' + accNum + '\_' + amount #NEEDS PROPER FORMATTING STILL

else:

print "Please enter a valid account number."

return transactionInfo

def deposit(self):

accNumInput = True

while (accNumInput):

accNum = raw\_input('Account Number: ')

#CHECK TO SEE IF VALID ACCOUNT NUMBER

if (1 == 1): #if account num is valid

amt = True

accNumInput = False

while (amt):

amount = int(input('Deposit Amount: '))

amount = amount\*100

if (amount > 999999):

print "Please enter a valid amount."

elif (amount < 0):

print "Please enter a valid amount."

else:

amt = False

#CREATE STRING TO WRITE TO FILE

accNum = str(accNum)

amount = str(amount)

transactionInfo = '01\_' + accNum + '\_' + amount #NEEDS PROPER FORMATTING STILL

else:

print "Please enter a valid account number."

return transactionInfo

def transfer(self):

accNumInput = True

accNumInput2 = True

while(accNumInput):

accNumTo = raw\_input('To Account Number: ')

#CHECK to SEE IF FIRST ACCOUNT NUMBER IS VALID

if (1 == 1):

while (accNumInput2):

accNumFrom = raw\_input('From Account Number: ')

#CHECK TO SEE IF SECOND ACCOUNT NUMBER IS VALID

if (1 == 1):

accNumInput = False

accNumInput2 = False

amt = True

while (amt):

amount = int(raw\_input('Transfer Amount: '))

amount = amount\*100

if (amount > 999999):

print "Please enter a valid transfer amount."

elif (amount < 0):

print "Please enter a valid transfer amount."

else:

amt = False

#create string for write file

accNumTo = str(accNumTo)

accNumFrom = str(accNumFrom)

amount = str(amount)

transactionInfo = '03\_' + accNumTo + '\_' + accNumFrom + '\_' + amount

else:

print "Please enter a valid account number."

else:

print "Please enter a valid account number."

return transactionInfo

def create(self):

accNumInput = True

accNameInput = True

while (accNumInput):

accNum = int(input('Enter your desired account number: '))

#Account Number must be 6 digits. Maximum of 999999, so if < 1000000, account number is 6 digits long

#CHECK TO SEE IF INPUT ACCOUNT NUMBER DOES NOT EXIST

if (1 == 1):

accNumInput = False

while (accNameInput):

accName = raw\_input('Enter your desired account name: ')

if (len(accName) > 15 | len(accName) == 0):

print "Please enter a valid account name."

else:

#create account number here

accNameInput = False

#create string for write file

accNum = str(accNum)

accName = str(accName)

transactionInfo = '04\_' + accNum + "\_" + accName #proper formatting on end of string is needed

else:

print "Please enter a valid account number."

return transactionInfo

def delete(self):

accNumInput = True

accNameInput = True

while (accNumInput):

accNum = int(input('Enter the account number: '))

#CHECK TO SEE IF INPUT ACCOUNT NUMBER EXISTS

if (1 == 1):

accNumInput = False

while (accNameInput):

accName = raw\_input('Enter the account name: ')

#CHECK TO SEE IF INPUT ACCOUNT NAME MATCHES ACCOUNT NUMBER

if (1 == 0):

print "Please enter the proper account name for this account."

else:

#delete account now

accNameInput = False

#create string for write file

accNum = str(accNum)

accName = str(accName)

transactionInfo = '05\_' + accNum + '\_' + accName #proper formatting on end of string is needed

else:

print "Please enter a valid account number."

return transactionInfo

#METHOD WHICH RUNS ANY TRANSACTIONS FOR A RETAIL DAY

#WILL WRITE ANY TRANSACTIONS TO FILE

#LOGOUT IS ACCEPTED AT THIS STAGE

def runAgentDay(self):

running = True

#CREATES TRANSACTION SUMMARY FILE

ts = time.time()

st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

save\_path = './TransactionSummaryFiles/'

file = 'Transaction\_Summary\_File\_\_' + st + '.txt'

filename = file.replace(":", "\_")

completeName = os.path.join(save\_path, filename)

f = open(completeName,'w')

while (running):

#STARTS ACCEPTING RETAIL TRANSACTIONS

transaction = raw\_input('Perform a transaction: ')

transaction.lower()

#TESTS INPUT FOR WHICH TRANSACTION TYPE TO PERFORM

if (transaction == "withdraw"):

newTrans = self.withdraw()

f.write(newTrans + '\n')

elif (transaction == "deposit"):

newTrans = self.deposit()

f.write(newTrans + '\n')

elif (transaction == "transfer"):

newTrans = self.transfer()

f.write(newTrans + '\n')

elif (transaction == "create"):

newTrans = self.create()

f.write(newTrans + '\n')

elif (transaction == "delete"):

newTrans = self.delete()

f.write(newTrans + '\n')

elif (transaction == "logout"):

f.close()

running = False

else:

print "Please enter a valid transaction type."

return False

###########################################################################################################

def openBankingSystem():

loggedIn = True

while (loggedIn):

#GETS LOGIN TO START, STAGE 0

firstInput = raw\_input('Type "login" to login: ')

firstInput.lower()

if (firstInput == "login"):

pickDay = True

while (pickDay):

#ACCEPTS INPUT FOR AGENT OR RETAIL, STAGE 1

dayType = raw\_input('agent or retail: ')

dayType.lower()

if (dayType == "retail"):

pickDay = False

retailDay = retail(dayType,0)

loggedIn = retailDay.runRetailDay()

elif (dayType == "agent"):

pickDay = False

agentDay = agent(dayType)

loggedIn = agentDay.runAgentDay()

else:

print "Please enter a valid input.\n"

else:

print "Please enter a valid input.\n"

#STARTS OVER AGAIN AFTER LOGOUT AT STAGE 0

return openBankingSystem()

###### MAIN PROGRAM ######

#open current accounts file

openBankingSystem()

#close curren accounts file

#ERROR AND TODO LOG

#wont accept any whitespace on string

#methods in each agent and retail

#code needs comments and variable names may need work