Bank Emulation Program Documentation v1.0.0

Isaiah Harshbarger – idh5029@psu.edu – 9 0078 6731

Louden Maclay – lrm5407@psu.edu – 9 6056 3761

Lucas Overmoyer – email – ID#

Summary

Our bank emulation program is meant to imitate the functionality of a real-world bank by simulating human conversation, various customer-bank interactions, and allowing financial plays something like they would go in real life. After instantiation, the initialization process will begin, which sets up an actor’s profile and then allows them to perform actions. In our bank environment, there are four main players that work together to simulate many of the workings of an actual bank: customer, manager, advisor, and teller. For example, customers can make checking account changes or inquiries by either “visiting” an ATM or “talking” to a teller. Tellers can also help people manage their business with the bank, advisors can help customers with their financial planning, and managers have administrative powers, such as the ability to view bank information and sensitive customer information. Overall, the different objects in our program work together to provide functionality that much resembles the various interactions, exchanges, and conversations that occur millions of times daily at banks across the world.

Visual Representation of Classes

Person Class:

The Person class contains an initialization function which prompts the user for their full name, birthdate, and address, with checks inserted to verify valid entries. It also produces and assigns a fake, randomly-generated social security number for the person.

It contains two class attributes (customer\_list and past\_customer\_list). At instantiation, customers are added to customer\_list. Upon deletion by a manager, they are moved to the past\_customer\_list.

**def talk(self):**

* Abstract class method, raises a NotImplementedError when talk hasn’t been explicitly defined in subclasses

**def make\_ssn():** (not a method of the person class but is called in its initialization)

* Uses the random python package to produce a fake social security number
* Returns a string in the form: ‘000-00-0000’

Customer Class:

The Customer class inherits from the Person class and uses its initialization as part of its own. The initialization for a customer is more involved than for an employee. The class attribute customer\_number, representing the number of customers created, is incremented by one before the instance variable customer\_number is set to the class attribute’s current value. Then, an account number is made by adding an arbitrary large number to the customer’s number for an authentic looking number. A made-up routing number is assigned. And all customer balances are set to start at 0. The customer is initialized as not deleted. The customer is then asked to set a pin before making their initial deposit. Customers are the financial players who can take out loans, make investments, and maintain a checking account.

**def \_\_mfAccountStatus(self):**

* Prints a message on the screen telling the user their mutual fund account balance, a reasonable, projected return rate for this investment, and their prospective balance after a year based on this return rate.

**def \_\_mgfAccountStatus(self):**

* Prints a message on the screen telling the user their precious metal and gem fund account balance, a reasonable, projected return rate for this investment, and their prospective balance after a year based on this return rate.

**def \_\_cryptoAccountStatus(self):**

* Prints a message on the screen telling the user their cryptocurrency account balance, a reasonable, projected return rate for this investment, and their prospective balance after a year based on this return rate.

**def \_\_str\_\_(self):**

* Overloads the string representation of a customer object to display their personal information in string format, displaying the customer’s name, date of birth, address, account and routing numbers, checking account balance, and net worth (checking account + investment accounts balances added together)

**def \_\_deposit(self):**

* Prompts user for an amount to deposit to their checking account, checking for a valid input.
* Prints the updated balance once the deposit is complete

**def \_\_withdrawal(self):**

* Calls \_\_check\_balance to display balance
* Prompts user for amount to withdrawal from their checking account, ensuring a valid input that is no more than his/her balance.s
* Prints the updated balance once the withdrawal is complete

**def \_\_check\_balance(self):**

* Prints the customer’s checking account balance

**def \_\_CheckOutstandingBalance(self):**

* Prints the customer’s outstanding balance with the bank from taking out a loan

**def atm(self):**

* Checks to make sure the customer accessing the ATM is an active customer. For deleted customers, prints that only active customers can use the ATM
* Prompts customer for their pin. If entered incorrectly 3 times, prints ‘too many failed attempts’ and the customer must use ATM method again to attempt to access.
* If access is granted, prompts the customer to choose (1)balance inquiry, (2)deposit, (3)withdrawal, or (4)exit.
  1. Calls \_\_check\_balance() then prompts user if they would like another transaction. Returns to menu if yes, exits if no.
  2. Calls \_\_deposit() then prompts user if they would like another transaction. Returns to menu if yes, exits if no.
  3. Asks the user if he/she will accept the $3 surcharge for ATM withdrawals. If yes, subtracts 3 from balance and calls \_\_withdrawal(), if not, then next step. Then prompts user if they would like another transaction. Returns to menu if yes, exits if no.
  4. Prints ‘Goodbye!’ and exits

**def talk(self):**

* Randomly prints one of two things that a customer might say.

Employee Class:

The employee class inherits from the person class and uses its initialization. This class provides some basic employee functions and is the parent class for each of the three types of employees.

**def ShowCustomerInfo(self, other):**

* If the customer is active (not a deleted customer), then this will print the customer’s name, date of birth, address, account and routing numbers, and their checking account balance.
* If the customer has been deleted by a manager, an inactive customer message is printed and no customer information is displayed.

**def DeleteCustomer(self, other):**

* When called by a non-manager, this function informs the employee who called it that they do not have the ability to perform this action.

**def SeeCustomers(self):**

* When called by a non-manager, this will just print just the number of active customers.

**def talk(self):**

* Employee introduces themselves but recommends anybody looking for help to ask a specialized employee.

Advisor Class:

The advisor class inherits from the employee class and uses the same initialization process. A financial advisor can help a customer take out a loan or get involved with any of the three offered investment opportunities.

**def \_\_offer\_loan(self, other):**

* Prompts the user for how much the customer would like to borrow. After ensuring valid input, the user enters for how long they’d like the loan term to be. Then, based on the customer’s wealth relative to the loan amount and the loan term length, an interest rate is generated using the make\_interest\_rate function.
* The proposed loan and attached interest rate are presented to the customer, who can either choose to accept or decline the loan. If accepted, the total interest to be paid over the course of the loan term is added to the customer’s outstanding balance with the bank.

**def make\_interest\_rate(amount, net\_worth, period):**

(not an advisor class method but is used in choosing the interest rate on a proposed loan)

* Returns the interest rate (Ex: 4.123) that will be offered to a customer based on the size of the proposed loan, their net worth, and the loan term length.
* The more risk the bank is perceived to incur (the greater the likelihood of the loan not being paid back), the greater the interest rate, for this bank. So, the greater the ratio of loan amount to customer’s net worth (checking account balance + all investment account balances), the greater the interest rate. Then, the greater the net worth, the less the interest rate. The longer the period, the less break a customer receives on their interest rate. This function combines these elements to yield an appropriate interest rate.

**def \_\_open\_investment\_account(self, other):**

* A customer must have $1,000 in checking to start investing, so if one does not, he/she is told to acquire more funds.
* If able, the customer can learn more about any of the investment options: mutual fund, precious metal & gem fund, or cryptocurrencies; or they can exit the investment menu.
* The customer is informed that from mutual fund, to precious metal & gem fund, to cryptocurrency, the safety of the investment decreases, but the potential reward increases.
* The customer can invest in any or all options and is asked to verify their investment after entering the amount but before finalizing. The amount invested is deducted from the customer’s checking account, provided the amount is there, and added to their respective investment account balance.

**def GiveAdvice(self, other):**

* This method is used for an advisor to give a customer the choice between discussing/taking out a loan and making/checking investments. The options are 1) Discuss a loan 2) Open/Add to investment account 3) Check performance of investment portfolio(s) 4) Exit the advising session.
  + Option 1) Calls self.\_\_offer\_loan(other) before asking if they would like to perform another action or exit advising
  + Option 2) Calls self.\_\_open\_investment\_account(other) before asking if they would like to perform another action or exit advising
  + Option 3) If all investment accounts are empty, such a message is printed. Otherwise, for each investment account with a balance, the appropriate customer method to display that account’s status is called.

**def talk(self):**

* The financial advisor introduces him/herself in one randomly chosen way out of three possibilities.

Manager Class:

The manager class inherits from the employee class and uses the same initialization process. A manager can perform more administrative tasks that have to do with bank management and oversight. Directly accessing a customer’s Social Security Number, deleting/reactivating customers, viewing a complete customer list, and seeing the total debts to the bank are things only managers can do.

**def ShowCustomerInfo(self, other):**

* If the referenced customer is inactive, the manager is informed of this.
* Otherwise, the customer’s name, birthdate, address, SSN, acct. #, routing #, checking acct. balance, and net worth are printed.

**def SeeCustomers(self):**

* This will print the name and account number of every active customer of the bank.

**def DeleteCustomer(self, other):**

* This “deletes” a customer by moving them from the current customer list to the past customer list. This disables a customer’s ability to use an ATM and get service from a teller or advice from an advisor, effectively deactivating the account.

**def ReactivateCustomer(self, other):**

* Restores a past customer’s full functionality as a customer with I.L.L & Sons by removing them from person class’s past customer list and adding them back to the active customer list.
* After reactivation,\_customer\_\_deleted\_customer is set back to False and customers can again access the ATM, advisor, teller, etc.

**def SeeDebtsToBank(self):**

* This will print a statement that indicates the total of the outstanding balances owed to the bank (the total amount loaned out plus interest) and how many customers there are with outstanding balances.

**def talk(self):**

* The manager introduces him/herself by name and as a manager at I.L.L & Sons.

Teller Class:

The teller class inherits from the employee class and uses the same initialization process. A teller can help a customer perform basic actions he/she wouldn’t directly be able to perform themselves or without an ATM but will not give advice or be able to help with their financial planning. Theoretically, tellers recognize customers by face, so they require no authentication to perform these actions.

**def \_\_TransferFromInvestmentAccount(self, other):**

* If the customer has made no investments, then a message will be printed telling them so.
* Otherwise, for each of their investment accounts with a balance, they will be prompted to accept or decline a 5% fee for transfers from an investment account to their checking account. If they accept the fee, they can enter a valid amount to transfer from that investment account to their checking account. 95% of the entered amount will move to their checking account and the rest is lost to the bank.

**def \_\_PaymentOnOutstandingBalance(self, other):**

* The customer’s outstanding balance is printed.
* If the customer has no outstanding balance, they are congratulated on their financial responsibility.
* If they have no funds to pay off their outstanding balance, they are told to return when they can make a payment.
* Otherwise, the customer is prompted to enter an amount to pay off their outstanding balance. This amount is checked for validity by being no more than the customer’s checking account balance or outstanding balance. This amount is deducted from the customer’s checking account and put towards paying off their outstanding balance.

**def ServeCustomer(self, other):**

* This presents the user with 7 choices of things a teller could do for a customer. Checks for a valid choice before doing one of the following:
  + Choice 1) Check customer balance. This calls \_customer\_\_check\_balance() and then asks if that will be all for today, responding accordingly, either exiting or again displaying the menu.
  + Choice 2) Make a deposit to customer’s checking account. This calls \_customer\_\_deposit() and then asks if that will be all for today, responding accordingly.
  + Choice 3) Make a withdrawal from customer’s checking account. This calls \_customer\_\_withdrawal() and then asks if that will be all for today, responding accordingly.
  + Choice 4) Transfer funds from an investment account to customer’s checking account. Calls \_\_TransferFromInvestmentAccount(other) and then asks if that will be all for today, responding accordingly.
  + Choice 5) Make a payment on customer’s outstanding balance. Calls \_\_PaymentOnOutstandingBalance(other) and then asks if that will be all for today, responding accordingly.
  + Choice 6) Reminds customer of their pin so they can access the ATM feature. Prints a statement with customer’s pin and reminds the customer to write it down and then asks if that will be all for today, responding accordingly**.**

**def talk(self):**

* The teller introduces him/herself by name and as a teller at I.L.L & Sons in one randomly selected way out of three possibilities.