Bank Emulation Program Documentation v1.0.0

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

Isaiah Harshbarger – email – ID#

Lucas Overmoyer – email – ID#

Summary

The bank emulation program contains various methods, classes, and functions that in unison, imitate the functionality of a real-world bank. Users may begin by assigning person-objects, such as customer, manager, advisor, or teller, to a variable. After the assignment, an instantiation process will begin which prompts the user for different information and stores it, depending on the type of person-object being created. Each person-object has access to numerous methods; some of these take another person-object as an argument to simulate human to human interaction within a bank, and others simulate simpler actions such as introducing one’s self or viewing a list of customers. However, not every person-object has access to every method. For example, only the manager-object has direct access to methods capable of viewing sensitive information or making significant changes. In short, this program provides functionalities, from depositing money to investing in cryptocurrencies, which accurately mimics those observed in a bank.

Visual Representation of Classes

Person Class:

The Person class contains an initialization function which, when called, prompts the user for their first name, middle name, last name, birthdate, and address. It also produces and assigns a faux social security number for the person and stores it, along with the other information collected.

It contains two class attributes (customer\_list and past\_customer\_list) which are utilized by subclasses to keep an updated list of current and past customers.

**def make\_ssn():** (not a method belonging to the class, but created to be used in the init function)

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

**def talk(self):**

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

Customer Class:

The Customer class inherits from the Person class and uses the init function from Person in its own init function. In addition, it initializes an account number, routing number, balance, customer number, and investment account balances. It then increments the class variable customer\_number by one, sets up the account number for the next instance created, appends itself to the Person’s class variable customer\_list, sets \_\_deleted\_customer to false, prompts the user to create a 4-digit pin, and prompts the user for their initial deposit.

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

* Uses random package to create a reasonable, random return rate for a mutual fund
* Prints a message on the screen telling the user their mutual fund account balance, expected return rate, and expected balance after a year.

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

* Uses random package to create a reasonable, random return rate for a precious metal and gem fund
* Prints a message on the screen telling the user their precious metal and gem fund account balance, expected return rate, and expected balance after a year.

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

* Uses random package to create a reasonable, random return rate for a cryptocurrency investment
* Prints a message on the screen telling the user their cryptocurrency investment account balance, expected return rate, and expected balance after a year.

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

* Overloads the string representation of a customer object to display their personal information in string format

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

* Prompts user for an amount to deposit
* Prints out when the deposit is complete as well as the new balance

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

* Calls \_\_check\_balance to display balance
* Prompts user for amount to withdrawal
* Prints out when the withdrawal is complete as well as the new balance

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

* Prints the balance of customer’s checking account

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

* Prints a customer’s outstanding balance with the bank

**def atm(self):**

* Checks to make sure the customer accessing the ATM is a current customer, if not, prints that only active customers can use the ATM
* Prompts customer for their pin, if entered incorrectly 4 times prints ‘too many failed attempts’ and the customer must use ATM method again to access it
* 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
  2. Calls \_\_deposit() then prompts user if they would like another transaction
  3. Asks the user if he/she will accept the $3 surcharge for ATM withdrawals(skips next step if no), if yes subtracts 3 from balance and calls \_\_withdrawal() then prompts user if they would like another transaction.
  4. Prints ‘Goodbye!’ and exits

**def talk(self):**

* Prints: “Hello! I’m {customer’s first name}”

Advisor Class:

Employee Class:

Manager Class:

Teller Class: