Requirement Specifications:

In this banking program the user should have multiple options regarding their ability to access/manage their money, including: Adding Accounts, Accessing those Accounts, making Withdrawals/Deposits and Quitting.

* When adding the account, the user will input their name and type of account. If the user is new, more information will be required. This includes their address, telephone number, age and type of customer. If the user already has an account/made an account successfully then an account id will be issued to the user. If there was an error in creating the account, a failed memo will be returned.
* When the user accesses their accounts, they will be asked their name. If the name isn’t recognized, the user will be asked if they wish to continue the banking app. If the name is recognized, the accounts associated with the user will be returned.
* When the user selects to make a withdrawal/deposit, they will be asked for their account ID. If the ID is recognized the user will be asked the amount to be withdrawn or deposited. After user input, the final balance will be updated and returned to the user.
* When the user selects to quit, the application will shut down.
* There will also be a section that will track the current interest on an individual’s account as well as their total transactions including: transaction type, amount, fees.
* Both the savings and checking accounts will acquired interest which will also be kept track of throughout the app’s use.

Cases for all Scenarios:

**Initial Directory**:

|  |  |
| --- | --- |
| User Actions | System Response |
| 1. User selects a specific command    1. Add Account    2. Accessing Account    3. Withdrawal/Deposit    4. Quit |  |
|  | System proceeds to the specified command.  Invalid input: system re-outputs initial menu. |

**Add Account**:

|  |  |
| --- | --- |
| User Actions | System Response |
| 1. User enters in name |  |
|  | System asks what type of account the user has. |
| 1. User inputs account type: Savings/Checking |  |
|  | If user is new user will be prompted for more information. |
| 1. User inputs address, telephone, age, type of customer. |  |
|  | New account is created/account already exists.  ID is returned to the user |
|  | If Add Account fails, user is prompted as such. |

**List Accounts**:

|  |  |
| --- | --- |
| User Actions | System Response |
| 1. User inputs name |  |
|  | If name isn’t recognized, system will ask user to return to main menu or quit. |
|  | If name is recognized, accounts of that user are returned. |

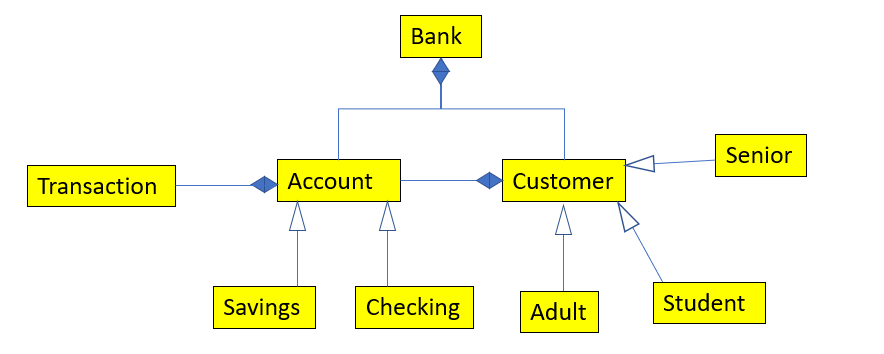
**Withdrawal/Deposits**:

|  |  |
| --- | --- |
| User Actions | System Response |
| 1. User inputs account ID |  |
|  | If account ID isn’t recognized, then the system will ask user to return to main menu or quit. |
|  | If account ID is recognized, the amount to withdrawal/deposit will be asked |
| 1. User inputs amount to be withdrawn/deposited. |  |
|  | System updates users current balance. |

**Quit**:

|  |  |
| --- | --- |
| User Actions | System Response |
| 1. User inputs specified quit key |  |
|  | System closes, ending banking app. |

UML Diagram:



**Bank**:

-int account\_id

-int customer\_id

-vector<Account\*> accounts

-vector<Customer\*> customers

-Vector<int> find\_accounts\_by\_name (string name)

-Customer\* find\_customer (string name)

-Account\* add\_account (Customer\* cust, string account\_type)

+Bank() : account\_id(1000), customer\_id(1000)

+Account\* add\_account (string name, string account\_type)

+Account\* add\_account (string name, string address, string telephone, int age, string cust\_type, string accout\_type)

+void make\_deposit (int acct\_number, double amt)

+void make\_withdrawal (int acct\_number, double amt)

+vector<int> get\_accounts (string name)

+Account\* get\_account (int acct\_number)

**Account**:

-Customer\* customer

-double balance

-int account\_number

-vector<Transaction\*> transactions  
-string get\_fees()

-void add\_interest (double interest)

+Account (Customer\* cust, int id) : customer(cust)

+Customer\* get\_customer()

+void set\_customer (Customer\* cust)

+int get\_account()

+void set\_balance (double new\_balance)

+void set\_balance (int account\_number)

+double get\_balance()

+virtual string to\_string()

+virtual void deposit (double amt)

+virtual void withdrawal (double amt)

+virtual void add\_interest

**Customer**:

-string name

-int age

-string telephone

-string address

-string cust\_type

-int id

+Customer (string name, int id, string cust\_type)

+string get\_name()

+int get\_id ()

+string get\_cust\_type()

+id get\_age()

+string get\_telephone()

+string get\_address()

+void set\_name (string name)

+void set\_age (int age)

+void set\_telephone (string telephone)

+void set\_address (string address)  
+void set\_cust\_type (string cut\_type)

**Transaction**:

-string customer\_number

-string transaction\_type

-double amount

-string fees

+Transaction (int customer\_number, string type, double amt, string fees)

+string process\_tran()

**Class Savings\_Account**:

+void deposit (double amt)

+void withdraw (double amt)

+void add\_interest()

**Checking\_Account**:

+void deposit (double amt)

+void withdraw (double amt)

+void add\_interest()

**Class Senior**:

-const int SAVINGS-INTEREST

-const int CHECK\_INTEREST

-const int CHECK\_CHARGE

-const int OVER\_DRAFT

**Class Adult**:

-const int SAVINGS-INTEREST

-const int CHECK\_INTEREST

-const int CHECK\_CHARGE

-const int OVER\_DRAFT

**Class Student**:

-const int SAVINGS-INTEREST

-const int CHECK\_INTEREST

-const int CHECK\_CHARGE

-const int OVER\_DRAFT

Pseudocode:

**Add\_Account in Banking\_Application.cpp**

1. System prompts user to input their name and account type
   1. Account type is stored in a variable
   2. Prompts the user they will need to add more information
2. Create an account by calling add\_account
3. Required inputs (for new user)
   1. Address
   2. Telephone
   3. Age
   4. Type of Customer
4. System returns account ID for the new user by passing the previously entered information into bank.add\_account.
   1. If failed, system will notify as such.

**Make\_deposit() methods in Bank.h and Banking\_Application.cpp**

Bank.h

1. System checks if there is a valid account/account number for the user
   1. If not return to menu and prompt to create account by add\_account
2. System prompts user for deposit amount
   1. System removes the amount from the account balance by make\_deposit

Banking\_Application.cpp

1. System prompts for user ID
   1. ID is not verified
2. System prompts for deposit amount
   1. System removes the amount from the account balance virtual void deposit(double atm)

**Make\_withdrawal() in Banking\_Application.cpp**

1. System prompts for user ID
   1. ID is not verified
2. System prompts for withdrawal amount
   1. System adds the amount from the account balance by calling virtual void withdraw(double atm)

**Overloaded add\_account() methods in Bank.h**

1. Create a new object of Customer class
2. Prompt user for
   1. Name
   2. Address
   3. Telephone
   4. Age
   5. Customer type
   6. Account type
3. Designate the object type based off the customer type
   1. Adult
   2. Senior
   3. Student
4. Call add\_account with customers name and account type (passing info in) and verify if account creation was successful

**Get\_account() in the Bank.h**

1. Pass in the account number
2. Loop through the accounts vector
3. If
   1. Account related to the account number is found
4. Return that account
5. Return NULL if account isn’t found

Bank Data Storage Description:

Each customer is assigned a specific ID, the first customer at the bank being assigned 0001, and incrementing from there. There is a vector that holds both customers and accounts. These vectors allow access to an individual customer/account. These accounts and customers are linked through a specific ID (mentioned above) and the users name. When a user wishes to access their account, by inputting their information the system will loop through the current vectors until a matching one is found, or the user is asked to create a new account because they don’t have one at the bank.

Transactions will be accessed by the users account ID and name. This provides access to the user’s account/balance/options. From here the User can access the withdrawal/deposit page and alter their current balance in the bank.