The software for the banking application should be able to let the bank client (user) interact with the program and make decisions. For each client, the program should store personal information such as their name, age, address and phone number. The software should also be able to store multiple accounts, specifying if each account is a checking or savings account, as well as who the account belongs to. For each account, the program should allow the user to store or withdraw money from either of their accounts, notifying the user of any fees for money withdrawal. There should also, for each account, be a record of transactions, showing when money has come in and out of the account. The program should implement the correct interest rate for each specific kind of account.

2.

* Use Case for New User adding an Account

|  |  |
| --- | --- |
| User’s Action | System’s Response |
| User issues command to add an account |  |
|  | System prompts user to input their name |
| User inputs their name |  |
|  | System prompts user for type of account |
| User inputs command corresponding to desired account type |  |
|  | System notifies user that more information is needed and prompts user for address |
| User inputs address |  |
|  | System prompts user for age |
| User inputs age |  |
|  | System prompts user to designate what kind of customer they are |
| User inputs command corresponding to customer type |  |
|  | System creates the desired account and outputs new account ID. If creation fails, outputs message detailing failure to user |

* Use Case listing Accounts

|  |  |
| --- | --- |
| User’s Action | System’s Response |
| User inputs command to list accounts |  |
|  | System prompts user for their name |
| User inputs name |  |
|  | System displays account ID for each account belonging to designated user along with total number of accounts |

* Use Case for making a deposit

|  |  |
| --- | --- |
|  |  |
| User inputs command to make a deposit |  |
|  | System prompts user for account ID |
| User inputs account ID of desired account |  |
|  | System prompts user for desired amount to deposit |
| User inputs desired deposit amount |  |
|  | System deposits desired amount into desired account |

* Use Case for making a withdrawl

|  |  |
| --- | --- |
|  |  |
| User inputs command to make a withdrawl |  |
|  | System prompts user for account ID |
| User inputs account ID of desired account |  |
|  | System prompts user for desired amount to withdraw |
| User inputs desired withdraw amount |  |
|  | System withdraws desired amount from desired account |

3.

Transaction

Checking

Student

Adult

Savings

Account

Customer

Bank

Senior

4.

* Add\_Account
  + Prompt user for their name
  + Store name
  + Prompt user for designation for account type
  + If “savings” is designated
    - Store “savings”
  + Or
    - store “checking”
  + Attempt to create a pointer to the new account
  + If pointer points to NULL
    - Prompt user for address
    - Store address
    - Prompt user for phone number
    - Store number
    - Prompt user for age
    - Store age
    - Prompt user to select customer type from list
    - Store designated type
  + If the account was created
    - Display message giving user new account ID
  + or
    - saying the account creation failed
* Make\_deposit(Bank &bank)
  + Prompt user for account ID
  + Store given ID
  + Prompt user for desired amount
  + Store given amount
  + Access the accounts stored in bank parameter
  + Call make\_deposit(int acct\_number, double amt)
    - Search the bank for the account with acct\_number
    - If the account exists
      * Add given amount to account
    - Else
      * Display message saying the account doesn’t exist
* Make\_withdrawl(Bank &bank)
  + Prompt user for account ID
  + Store given ID
  + Prompt user for desired amount
  + Store given amount
  + Access the accounts stored in bank parameter
  + Call make\_withdrawl(int acct\_number, double amt)
    - Search the bank for the account with acct\_number
    - If the account exists
      * If the amount is less than or equal to amount in account
        + Take given amount out of account
      * Else
        + Display message saying the account does not have enough money
    - Else
      * Display message saying the account doesn’t exist
* Add\_account(string name, string account\_type)
  + Search for a customer with the given name
  + If the customer doesn’t exist
    - Return NULL value
  + Call add\_account(cust, account\_type)
    - Create a new Account object with the designated customer and account type
* Get\_account(int acct\_number)
  + For each account that has been created
    - If the account number at that index matches the parameter
      * Return the account number at that index
    - Otherwise, return NULL value

5. The account IDs of each account and the customer IDs for each unique customer will be taken from the respective Bank class counter, which will be incremented after each new account or customer is created. The account IDs will be stored in the account object and the customer IDs will be stored in the customer object. The accounts will be linked to the customer by storing the name of the customer who owns the account. Each account will have a vector that stores transactions involving that account, so the transactions are linked to the customer owning the account.