1. The software should be able to simulate the operation of normal bank, that includes adding a new customer, adding accounts for customers, making transaction from those accounts, deposit, withdraw money.

* These customers/accounts should be able to store basic information of customers, the money in the account and the history of transaction
* When run, the program should prompt the user which action to do, ask for input for the information, and save it to a file.

1. Use case:

* Add account:

|  |  |  |
| --- | --- | --- |
| Step | User’s action | System’s response |
| 1. | User issues the command to add account |  |
| 2. |  | System prompts for name |
| 3. | User enter name |  |
| 4. |  | Systems prompts for type of accounts |
| 5. | User enter the command for type of account |  |
| 6. |  | System will check if user is new or not, if new, the user will be asked for additional information. These information will then be store |

* List account:

|  |  |  |
| --- | --- | --- |
| Step | User’s action | System’s response |
| 1. | User issues the command to list accounts |  |
| 2. |  | System prompts for the name |
| 3. | User enter name |  |
| 4. |  | System list all the account with the input name |

* Withdraw:

|  |  |  |
| --- | --- | --- |
| Step | User’s action | System’s response |
| 1. | User issues the command to make withdrawal |  |
| 2. |  | System prompts for the account ID |
| 3. | User enter account ID |  |
| 4. |  | System prompts for the amount to withdraw |
| 5. | User enter amount to withdraw |  |
| 6. |  | Systems prompts if successful/not, the amount of money left |

1. A close up of a logo

   Description automatically generatedUML diagram:
2. Pseudocode:

* Add\_account:
  + Prompt the user for name
  + Read the name
  + Output all the option for account type, prompt the user for account type
  + Read the account type
  + Pass the previous information to the add\_account function in bank object,
  + **If** user does not exist, function return NULL
  + **Then** prompt the user for address, telephone number, age, type of customer.
  + Read the input from the user
  + Attempt to create a account as a new user with all the information
  + **If** function return true, output the new account ID
  + **If** false, prompt the user that the account creating process is failed
* Make\_deposit():
  + Prompt the user for account ID
  + Read the account ID
  + Prompt the user for the deposit amount
  + Read the amount
  + Call the make\_deposit function in bank class
  + If the make\_deposit() function can get account with the ID, add the amount of money deposited
  + If the make\_deposit() function fail to get account information with the ID, return error and exit function.
* Make\_withdraw:
  + Prompt the user for account ID
  + Read the account ID
  + Prompt the user for the withdraw amount
  + Read the amount
  + Call the make\_withdraw function in the bank class with the previous information.
* Overloaded add\_account (2 values):
  + Find customer based on passed in name
  + If not found, return false.
  + If found, add a new account to bank’s customer vector
* Overloaded add\_account (6 values):
  + Create a new customer based on information passed into the function
  + Add a new account to the bank’s customers vector and return that account
* Get\_account:
  + Read through all the account, return an account with the same acc\_number as the acc\_number passed into the function

1. – The account number and the customer id counter will be create in the bank constructor, this counter will then be increment every time a new customer or account is created, and then used to create unique ID.

* Every account store pointer to a customer object, this will be the owner of that account
* Every account will have a vector of pointer to transaction class. These are the transactions made by accounts of customers. From a customer, we can find all the accounts and find all the transactions in those accounts.