Bao Tran

CS 273

Pete Tucker

HW05

The software will simulate the operations of a bank. The software should be able to store account information and identification of its members. As well, the software should also store and categorize the accounts based on age and special cases ( Senior, adult, student). The software should also store the name, address, age, and telephone number per customer account. For each customer, they should be able to make transactions and check balances from either checking or savings accounts. This would include depositing and withdrawing. As well, there should be a feature in which penalties are given if there is overdraft or insufficient funds. The software should also allow the bank to create new accounts and manage existing ones such as depositing and withdrawing directly from the account.

Use Cases

* Initial Use Case
* Bank Cases
  + Create new account/edit accounts
  + Get Checking
  + Get Savings
  + Get account
* Customer Cases
  + Withdrawal from Checking
  + Deposit to Checking
  + Withdrawal from Savings
  + Deposit to Savings
  + Get Checking
  + Get Savings

Initial use case

|  |  |  |
| --- | --- | --- |
| Step | User’s Action (Bank) | System’s Response |
| 1. | User Issues a command to the operating system to load and run the Banking Simulation program, specifying the file that contains the directory of account information |  |
| 2. |  | The Banking Simulation starts and a data file is loaded, initializing the accounts array. If a data file does not exist, an empty array for accounts is created |

Adding account use case

|  |  |  |
| --- | --- | --- |
| Step | User’s Action (Bank) | System’s Response |
| 1. | User Issues a command to add a new account |  |
| 2. |  | System Prompts for name |
| 3. | User enters name | If user cancels entry, process terminates. |
| 4. |  | System Prompts for address |
| 5. | User enters address | If user cancels entry, process terminates. |
| 6. |  | System Prompts for age |
| 7. | User enters age | If user cancels entry, process terminates. |
| 8. |  | System Prompts for telephone number |
| 9. | User enters telephone number | If user cancels entry, process terminates. |
| 10. |  | System Prompts for customer number |
| 11. | User enters customer number | If user cancels entry, process terminates. |
| 12. |  | The account array is updated to contain the new account. The user is prompt that adding the account was successful. |

Editing account use case

|  |  |  |
| --- | --- | --- |
| Step | User’s Action (Bank) | System’s Response |
| 1. | User Issues a command to edit existing account |  |
| 2. |  | System Prompts for customer number |
| 3. | User enters customer number | If user cancels entry, process terminates. |
| 4. |  | If the customer number is not found within data file, user is prompted that the customer does not exist and the process terminates. If a customer name is found, system prompts the user to select the data field to edit. |
| 5. | User enters which data field to edit | If user cancels entry, process terminates. If user enters invalid data field the system requests another user input |
| 6. |  | System prompts user to edit that data field |
| 7. | User edits data field | If user cancels entry, process terminates without changing the data field. |
| 8. |  | System changes the data field in the specified account. System Prompts if user would like to edit another data field for that customer number |
| 9. | User enters yes or no |  |
| 10. |  | If yes, system repeats steps 3 – 10. If no the process terminates |

Transaction use case.

|  |  |  |
| --- | --- | --- |
| Step | User’s Action (customer) | System’s Response |
| 1. | User issues a command to create a transaction |  |
| 2. |  | System prompts user to select a type of transaction (deposit or withdrawal) |
| 3. | User enters the type of transaction | If user cancels entry, process terminates. If entry is invalid system prompts to re-enter input. |
| 4. |  | System prompts user to select a which account to do the transaction (checking or saving) |
| 5. | User enters the specified account | If user cancels entry, process terminates. If entry is invalid system prompts to re-enter input. |
| 6. |  | System prompts the amount to withdrawal or deposit |
| 7. | User enters an amount | If user cancels entry, process terminates. If entry is invalid or if entry is out of the range checked by the system, the system prompts to re-enter input. |
| 8. |  | System adds or subtracts amount from the specified account. |
| 9. |  | System prompt to do another transaction or not. |
| 10. | User enters yes or no. | If user cancels entry, process terminates. If entry is invalid system prompts to re-enter input. |
| 11. |  | System repeats steps 2-10 if yes. If no the process terminates |

Check balances use case

|  |  |  |
| --- | --- | --- |
| Step | User’s Action (customer) | System’s Response |
| 1. | User issues a command to check balances |  |
| 2. |  | System prompts the user to input which account the user would like to see the balance of. (checking or savings) |
| 3. | User enters the account. | If user cancels entry, process terminates. If entry is invalid system prompts to re-enter input. |
| 4. |  | System will display the account type, balance, and transaction history. |
| 5. |  | System will prompt the user if they’d like to check another balance |
| 6. | User enters yes or no | If user cancels entry, process terminates. If entry is invalid system prompts to re-enter input. |
| 7. |  | System will repeat steps 2-6 if yes. If no the process will terminate. |

Add\_Account

1. Prompts user to enter name
2. Read input from user and store into name data field
3. Prompts user for account type (checking or savings)
4. Read input from user and store it in type data field
5. Checks if user is already in the data file
   1. If in array
      1. add another account index and copy address, phone number, and age of previous account.
      2. Display new account ID
   2. if not in array
      1. Add new account to bank’s array of accounts using the name and type data field
      2. Prompt user for address, telephone number, and age to input into the account’s data field
      3. Display new account ID

List\_Account

1. Prompts user to enter a name
2. Read input from user and store it in a temporary name data field
3. Searches through array of accounts for same name in name data field
   1. If name is the same
      1. Display account’s balance and ID
      2. Increment a counter
4. Display the total amount of accounts found under the given name

Make\_Deposit

1. Prompts user to enter account ID
2. Read user input into temporary ID data field
3. Prompt user to enter amount
4. Reader user input into temporary amount data field
5. Add amount user inputs into the account with the corresponding account ID from the account arrays

Make\_Withdrawal

1. Prompts user to enter account ID
2. Read user input into temporary ID data field
3. Prompt user to enter amount
4. Reader user input into temporary amount data field
5. subtracts amount user inputs into the account with the corresponding account ID from the account arrays

to\_string

1. returns a string containing information about the account’s owner, the account balance, and the account ID

deposit

1. changes the account balance by the specified amount.
2. Log the transaction into a transaction vector

withdraw

1. changes the account balance by the specified amount.
2. Log the transaction into a transaction vector

Find\_accounts\_by\_name

1. Create a vector to store accounts
2. For all accounts
   1. If account name = name passed through function
      1. Add account to account vector
3. Return account vector

\*find\_customer

1. For all customers in customer array
   1. If customer name = name passed through function
      1. return customer object
   2. else
      1. return NULL

\*add\_account

1. creates a new account for a customer object with an account type
2. returns the account

make\_deposit

1. finds the specified account with the account number
2. if the account exists
   1. deposit the specified amount into that account
3. else return NULL

make\_withdrawal

1. finds the specified account with the account number
2. if the account exists
   1. withdrawal the specified amount into that account
3. else return NULL

get\_account

1. for all accounts in account array
   1. if account number = specified account number
      1. return that account
   2. else return NULL

Bank Data Storage System:

In the banking system, there should be an array of accounts for all accounts held in the system. For every account, there should be a customer object that indicates the owner of the account as well as the customer information. As well, each account will have a vector of transaction objects that logs the transactions made in that account. Since the transactions are objects within the account objects, we can call get functions from the account object that will get information from the customer object. Using this method, we can also go through all the accounts and use get functions to return the account owner’s name. This allows us to find account belonging to a certain customer. Customer IDs should be the first letter of the first named followed by the last name followed by 8 randomly generated alphanumerical characters to avoid conflicts with customers with the same name. Account numbers should be comprised of 10 randomly generated alphanumerical characters to avoid conflicts. As well, both customer and account numbers should be checked if they already exist if it is a new customer or new account. If there is an existing customer, only a new account number will be generated that copies the customer ID.