Rachel Wofford

CS 273 HW 5

Design Document

Study the problem description and the code we have given you as a starting point, banking.zip on the class folder. You will need to develop the following items:

Part 1: A requirements specification of the software that needs to be created in your own words. This should be a short English narrative describing what you think the software should do and what it will operate like, not how it will implement its functions.

This software should operate as a program that a local bank will use. When the user opens the software, a menu will be displayed with five options: Add Account, List Accounts, Make Deposit, Make Withdrawal, and Quit.

If the user chooses Add Account, they will be prompted to input their name. If they already have an account, a new checking or savings account (user’s choice) with a new ID is created for that user. If a new user chooses Add Account, they will be prompted for additional information including address, phone number, and age. They will also indicate whether they are an adult, student, or senior customer and whether they want a checking or a savings account and a new account will be created for them.

If the user chooses List Accounts, they will be prompted to input their name. If the user already exists within the system, a list of their current accounts will be displayed. If the user does not already exist, no list will be output and the user will be informed that there was an error.

If the user chooses Make Deposit, they will be prompted to enter their account ID and the amount they wish to deposit. If the account ID points to a valid account, the amount of money the user indicates will be deposited in their account. If the account ID does not point to a valid account, the user will be informed that there was an error.

If the user chooses Make Withdrawal, the same process as Make Deposit will occur, money will just be withdrawn instead of deposited.

If the user chooses Quit, the program will end.

After the end of each menu option, other than Quit, the user will have the option to make another transaction until they decide to exit the program. Throughout the execution of this program, a list of the transactions that have been performed will be kept and stored with the account information. Additionally, both the checking and savings account will collect interest over time, which will be kept track of along with the other transactions.

2. Detailed use cases for all the scenarios you imagine the software will be used. Look at the phone directory example in the text book (section 1.5) to guide you with this.

Use Case: A list of the user actions and system responses for a particular sub-problem in the order that they are likely to occur.

Add Account for a pre-existing user:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to add account |  |
| 2 |  | Call Add\_Account function |
| 3 |  | Prompts user for name |
| 4 | Inputs name |  |
| 5 |  | Displays menu of account types and prompts user for desired account type |
| 6 | Inputs desired account type |  |
| 7 |  | Attempts to create an account |
| 8 |  | New account ID is output to user or user is informed that system failed to create an account for them |

Add Account for a new user:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to add account |  |
| 2 |  | Call Add\_Account function |
| 3 |  | Prompts user for name |
| 4 | Inputs name |  |
| 5 |  | Displays menu of account types and prompts user for desired account type |
| 6 | Inputs desired account type |  |
| 7 |  | Attempts to create an account |
| 8 |  | Attempt fails because user is new |
| 9 |  | Outputs to user that more information will be required and prompts user for address |
| 10 | Inputs address |  |
| 11 |  | Prompts user for telephone number |
| 12 | Inputs telephone number |  |
| 13 |  | Prompts user for age |
| 14 | Inputs age |  |
| 15 |  | Displays menu of customer types and prompts user for their customer type |
| 16 | Inputs customer type |  |
| 17 |  | Attempts to create an account |
| 18 |  | New account ID is output to user or user is informed that the system failed to create an account for them |

List Accounts:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to list accounts |  |
| 2 |  | Calls List\_Account function |
| 3 |  | Prompts user for name |
| 4 | Inputs name |  |
| 5 |  | Calls function get\_account based on user’s name input and assign it to a vector that will store all accounts that are a under a customer’s specified name |
| 6 |  | Loop through all accounts in the vector from step 5 |
| 7 |  | If the loop comes across an account that belongs to the specified user, it outputs that account information |
| 8 |  | Output the total number of accounts found for that user |

Make Deposit:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to make a deposit |  |
| 2 |  | Calls Make\_Deposit function |
| 3 |  | Prompts user to input account ID |
| 4 | Inputs account ID |  |
| 5 |  | Prompts user to input amount they wish to deposit |
| 6 | Inputs desired deposit amount |  |
| 7 |  | Calls make\_deposit function which accesses user’s account based on account ID |
| 8 |  | If an account with the given ID is found, the deposit is recorded in the account |

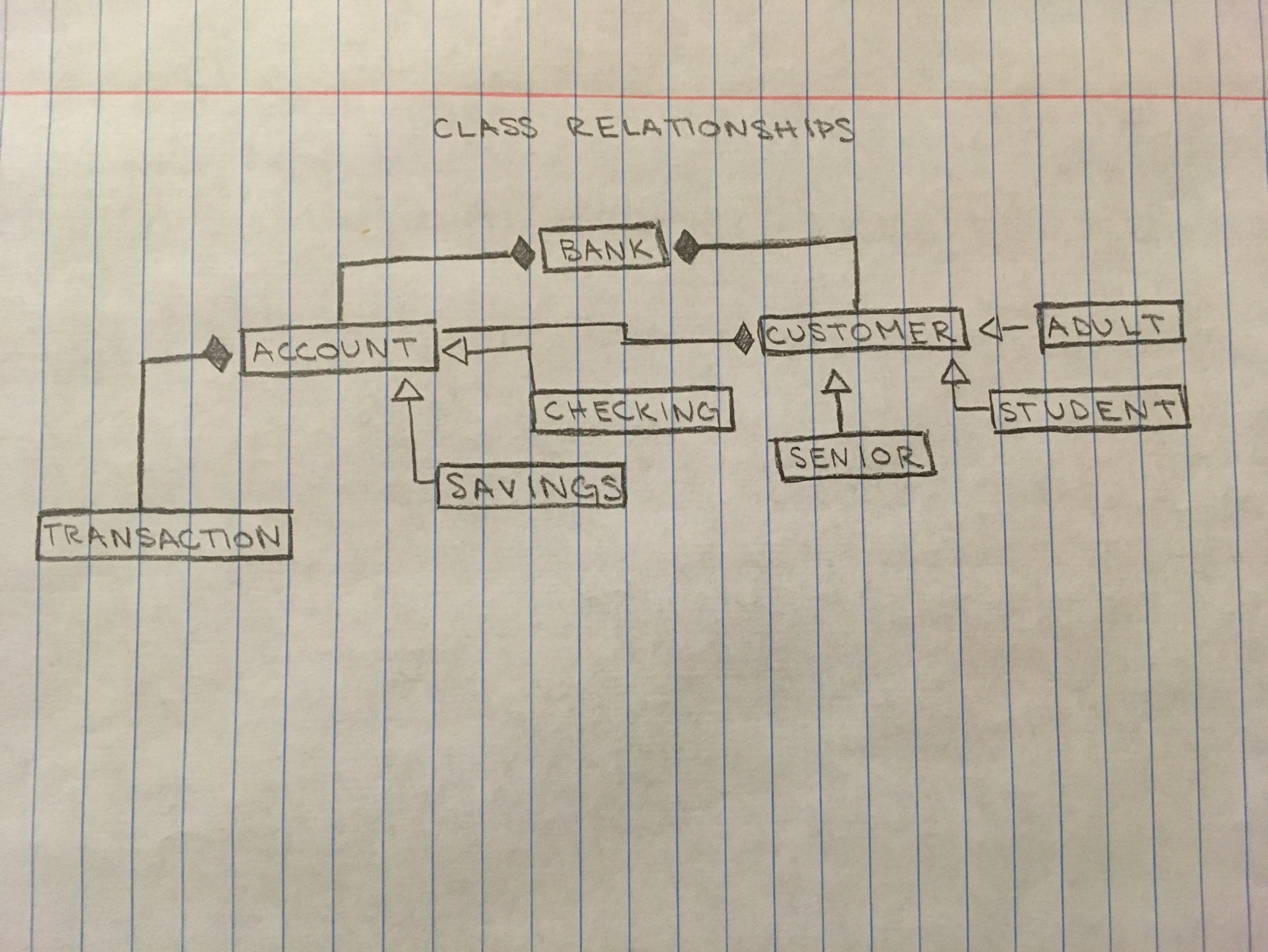
Make Withdrawal:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to make a withdrawal |  |
| 2 |  | Calls Make\_Withdrawal function |
| 3 |  | Prompts user to input account ID |
| 4 | Inputs account ID |  |
| 5 |  | Prompts user to input amount they wish to withdraw |
| 6 | Inputs desired withdrawal amount |  |
| 7 |  | Calls make\_withdrawal function which accesses user’s account based on account ID |
| 8 |  | If an account with the given ID is found, the withdrawal is recorded in the account |

Quit:

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Chooses option to quit |  |
| 2 |  | Outputs a goodbye statement |

3. UML diagrams to describe the relationship between the classes described in the problem. You do not need to describe any sequence diagrams unless you wish to.



|  |
| --- |
| Bank |
| -vector <Account \*> accounts  -vector <Customer \*> customers  -int account\_id  -int customer\_id |
| -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 account\_type)  +void make\_deposit(int acct\_number, double amt)  +void make\_withdrawal(int acct\_number, double amt)  +vector<int> get\_account(string name)  +Account \*get\_account(int acct\_number) |

|  |
| --- |
| Account |
| -double balance  -int account\_number  -vector<Transaction \*> transactions; |
| -string get\_fees()  -void add\_interest(double interest)  +Account(Customer \*cust, int id) : customer(cust), account\_number(id), balance(0)  +Customer \*get\_customer()  +void set\_customer(Customer \*cust)  +int get\_account()  +void set\_balance(double new\_balance)  +void set\_account(int account\_number)  +double get\_balance()  +virtual string to\_string()  +virtual void deposit(double amt)  +virtual void withdraw(double amt)  +virtual void add\_interest() |

|  |
| --- |
| Savings\_Account |
| + void deposit(double amt)  + void withdraw(double amt)  + void add\_interest(cons tint SAVINGS\_INTEREST) |

|  |
| --- |
| Checking\_Account |
| + void deposit(double amt)  + void withdraw(double amt)  + void add\_interest(const int CHECKING\_INTEREST) |

|  |
| --- |
| Transaction |
| - string customer\_number  - string transaction\_type  - double amount  -string fees |
| + Transaction (int customer\_number, string type, double amt, string fees)  + string process\_tran() |

|  |
| --- |
| Customer |
| - string name  - string address  - int age  - string telephone\_number  - int customer\_id  - int cust\_type |
| + Customer (int customer\_id, string name, string cust\_type)  + string get\_customer\_id()  + string get\_name()  + string get\_cust\_type()  + string get\_address()  + string get\_telephone\_number()  + int get\_age()  + void set\_name(string name)  + void set\_address(string address)  + void set\_age(int age)  + void set\_telephone\_number(string telephone\_number)  + void set\_cust\_type(string cust\_type) |

|  |
| --- |
| Senior |
| + const double SAVINGS\_INTEREST  + const double CHECK\_INTEREST  + const double CHECK\_CHARGE  + const double OVERDRAFT\_PENALTY |

|  |
| --- |
| Adult |
| + const double SAVINGS\_INTEREST  + const double CHECK\_INTEREST  + const double CHECK\_CHARGE  + const double OVERDRAFT\_PENALTY |

|  |
| --- |
| Student |
| + const double SAVINGS\_INTEREST  + const double CHECK\_INTEREST  + const double CHECK\_CHARGE  + const double OVERDRAFT\_PENALTY |

4. Pseudocode (see page 106 in your book for an example) to describe how the following methods will be implemented. Note, you will need to implement more code than is described here…

· Add\_Account in Banking\_Application.cpp,

1. Ask user to input their name
2. Accept name input from user
3. Display a menu with the type of account (checking or savings)
4. Accept user’s input for type of account
5. Store user’s choice for account type in a variable
6. Attempt to make an account by calling the add\_account function with the name and type of account the user input
7. If the function fails to create an account because the user is new:
   1. Tell the user that more information is necessary
   2. Accept user’s input for address, telephone number, age
   3. Display a menu to the user asking what type of customer they are (senior, student, adult)
   4. Accept user’s input for type of customer
   5. Attempt to make an account by calling the add\_account function with name, address, telephone number, age, customer type, and account type that the customer input
8. If an account is created in either step 6 or 7e, output the new account ID to the user
9. Else, tell the user that the system failed to create an account for them

· make\_deposit() methods in Bank.h and Banking\_Application.cpp,

1. Prompt user to input their account ID
2. Accept user’s input for account ID
3. Prompt user to input the amount they wish to deposit
4. Accept user’s input for deposit amount
5. Call make\_deposit function in Bank.h using user’s account ID and deposit amount
   1. Call get\_account function
   2. If an account with the given ID exists, withdraw the indicated amount from the account

· make\_withdrawal() in Banking\_Application.cpp

1. Prompt user to input their account ID
2. Accept user’s input for account ID
3. Prompt user to input the amount they wish to withdraw
4. Accept user’s input for withdrawal amount
5. Call make\_withdrawal function using user’s account ID and withdrawal amount
   1. Call get\_account function
   2. If an account with the given ID exists, withdraw the indicated amount from the account

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

1. Call this constructor when an Account object is created with parameters of name, address, telephone number, age, customer type, and account type
2. Create a pointer to a Customer object
3. Based on customer type passed into the function, create either an adult, senior, or student object
4. Add the adult, senior, or student object just created to the customers vector
5. Once the new customer object is created, call the add\_account function with the customer’s name and account type and return whether or not an account was able to be created for that user (true/false)

· get\_account() in the Bank.h

1. Call this function with acct\_number as the only parameter
2. For loop that searches through all of the pre-existing accounts:
   1. If the loop finds an account object that matches the account number, return the account object
3. If the for loop doesn’t find an account object that matches the account number, return NULL

5. Bank Data Storage Description – A description of HOW the account numbers for accounts and the customer id numbers will be generated and stored. How will accounts be linked to customers? How will transactions be linked to customers? (see Bank.h)

The account numbers for accounts and the customer id numbers will be generated in the same way. At the beginning of the main function of Banking\_Application.cpp, a Bank object is created that automatically generates a vector that can hold 1000 account IDs and a second vector that can hold 1000 customer IDs. Each customer will receive a unique ID number the first time they make an account. The very first customer will be assigned the ID number 0000. The value of the variable customer\_id will then increment and the next customer will be assigned the ID number 0001. This pattern will continue for all ID assignments. The same process occurs when account numbers are generated, just whenever a user makes an account instead of when there is a new user. Also, the variable account\_id will be incremented instead of customer\_id. The account numbers and customer IDs will be stored in the accounts and customers vectors created in the Bank class. The system will be able to access specific users or accounts from these vectors. Accounts will be linked to customers through a customer object that is created using the customer’s name, address, phone number, age, customer type and account type. When an account is created, a pointer to the customer that created the account will be made and the information will be stored. When a customer wants to access their account, they can input their customer ID and name and the system will loop through the vector of all accounts that have been made until it finds the indicated account or the user will be informed that the system failed to find the account. Transactions will be linked to customers in a similar way. For each transaction, a transaction object will be created. This object will be initialized with the customer’s ID number (and other parameters), which is how it will be linked to the customer that performed the transaction.