Grant Bingham

HW 5 - Polymorphism Part 1

Requirements Specification:

The project will require an umbrella class for the bank. This bank will be composed of multiple Accounts and Customers, and will be able to add and/ or modify said Accounts or Customers. Also, this bank will need to be able to search for a specific customer and return all the accounts that are associated with that customer. There will be two different types of accounts: Savings and Checking. For every Account created in the Bank, there will be a Customer associated with that account. Both types of accounts will allow a customer to deposit, withdraw, and see the balance of their accounts.

Each Customer will have certain information associated with them, such as their name, address, phone number, etc. However, there will be different types of customers, and each customer will have added functions to their account that will be specific to them.

Also, these Customers are all going to go out and make transactions at some point. For this reason, each account needs to be modified each time the customer associated with that account makes a transaction of some sort.

Lastly, here is what the main function will look like. First off, bank object will need to be created because there are several functions that are to be discussed that will take this bank object as a parameter. The main menu will first greet the customer, then present the customer with a list of options. The user may choose to add an account, see a list of their accounts, make a deposit/ withdrawal, or quit the program. If the user chooses to add an account, they will be asked for their name, and asked to select which types of account they would like to create, savings or checking. If the customer is new to the bank, the bank will prompt the user for their personal information so a new customer object can be created. If the user wants to see a list of their accounts, then all accounts associated with the customers name will be added to a vector, and the size of this vector will then be outputted which is the number of accounts associated with that customer. If the user chooses to make a deposit/ withdrawal, the user will be prompted for their account ID and the amount they wish to deposit/ withdraw. The balance of their account is then modified accordingly. If the user wants to quit, their session will end.

Cases/ Scenarios:

| User Actions | System Response |
| --- | --- |
| User creates Account with the bank. | An Account object is created and is added to the vector of Customer pointers in the Bank class. A Customer object is created with all the User’s information and is added to the vector of Customer pointers in the Bank class. The Account object is composed of the Customer pointer. A Checking Account object and a Savings Account object are both created and inherit from the Account object. Also, depending on who the user is, the program will create a Senior, Adult, or Student object which inherits from the customer object. |
| User makes a deposit. | The deposit() member is called from either the Checking or Savings account (depending on where the User want their money) and adds the amount deposited to the balance data member from the Account class. |
| User makes a withdrawal. | The withdrawal() member is called from either the Checking or Savings account (depending on where the User want their money) and subtracts the amount withdrawn from the balance data member from the Account class. |
| User makes a transaction. | A Transaction object is created with a specified transaction type and amount and is added to the pointer array that the Account class is composed of. The process\_tran() member function is called and subtracts the amount spent from the balance data member of the account class. |
| User searches for their entry and accounts associated with their entry. | The find\_accounts\_by\_name() function is called from the bank class. This function searches through the vector of customers for the name that was entered. One the customer is found, it then searches the vector of accounts for accounts that are associated with that customer. Each account that is associated with this customer is added to a separate empty vector that will be composed of this specific customer’s accounts. The function will return this vector. |

UML diagrams:

|  |
| --- |
| 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(sting 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 (Bank is composed of this class) |
| --- |
| 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), account\_number(id), balance(0)  double get\_balance()  Customer\* get\_customer()  to\_string()  int get\_account()  void set\_customer(Customer\* cust)  void set\_balance(double new\_balance)  void set\_account(int account\_number)  virtual string to\_string()  virtual void deposit(double amt)  virtual void withdrawal(double amt)  virtual void add\_interest() |

| Checking (Inherits from Account) |
| --- |
| void deposit()  void withdraw()  void add\_interest() |

| Savings (Inherits from Account) |
| --- |
| void deposit()  void withdraw()  void add\_interest() |

| Customer (Account and bank are composed of this class) |
| --- |
| string name  string address  int telephone\_number  int customer\_number |
|  |

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

Pseudocode:

Add\_Account:

* Prompts customer for name.
* Prompts user to select the type of account they would like to add (savings or checking).
* If the customer has already made accounts with the bank before, then the bank.add\_account() function is called and takes the customers name and account type as arguments.
* If it is the customer’s first time making an account, gather the user’s address, phone number, age, and whether they are an adult, senior, or student. All these pieces of info are stored into variables.
* The bank.add\_account() member is called from the bank class and takes all this previous variables as arguments.
* The function the outputs the new account ID

make\_deposit():

In the .cpp:

* Prompts user to enter the account ID of the account they would like to deposit to and also enter the amount that they would like to deposit.
* The bank.make\_deposit() is called and takes the account ID and amount as arguments.

In the .h

* Creates an Account pointer object and sets it equal to the specified account using the get\_account() function which takes the account number as an argument and returns the specified account.
* If the account exists, the deposit() function is called and takes the amount as an argument, and adds that amount to the current balance.
* A Transaction object is created and added to the vector of transactions.

make withdrawal():

In the .cpp:

* Prompts user to enter the account ID of the account they would like to withdraw from and also enter the amount that they would like to deposit.
* The bank.make\_withdrawal() is called and takes the account ID and amount as arguments.

In the .h

* Creates an Account pointer object and sets it equal to the specified account using the get\_account() function which takes the account number as an argument and returns the specified account.
* If the account exists, the withdraw() function is called and takes the amount as an argument, and subtracts that amount from the current balance.
* A Transaction object is created and added to the vector of transactions.

Overloaded add\_account():

* Takes the new customer’s information as arguments.
* Creates a Customer pointer object.
* Creates a Customer object of type Adult, Senior, or Student based on what was passed into the function.
* Assigns the pointer object to this object, and the pointer is added to the vector of Customer pointers.
* Returns the added account.

get\_account():

* Takes the account number as an argument.
* Searches the vector of account pointers for a match in account number.
* Once there is a match, the specified account is returned.

Bank Data Storage Description:

The account ID and customer ID numbers are part of the constructors for the Account and Customer class. Thus, a new account ID is assigned whenever am Account object is created, likewise for the customer ID. Also, the variables for account ID and Customer ID will be incremented every time an object is created to ensure that each object gets its own unique ID number.

In order to keep accounts linked to customers, the customer’s name will be a part of every account constructor. Transactions will be linked to customers the same way, only the transaction constructor will take the customer’s ID instead of their name.