Rawan Sarhan

Lab 5 Design

December 2, 2015

For Lab 5, I have 6 classes: Bank, Account, ApplyTransactions, CheckTransactions, Funds, BSTree. The Bank class deals with reading in the file into a queue, executing all the transactions from the queue, and printing out the open accounts. The Account class is for each of the accounts in the Bank. It keeps track of the first and last name and the account ID. This class also keeps track of all the funds and the history of transactions for each account. The Account class deals with the Funds class, with each account having an array of 10 funds (1 for each of the 10 funds an account can have). The Funds class is for keeping track of the balance in each of the funds for an account and the history of the single fund. The ApplyTransactions class is what actually executes each of the transactions. However, before the transactions are executed, the CheckTransactions class makes sure the transaction is valid and can be done. For example, the CheckTransactions class will check to make sure that there are no duplicates of accounts before opening a new account. The BSTree class has all the accounts in the bank and is placed in the private part of the Bank class. Traversing through the tree at the end for phase 3 will print out all the open accounts and their balances in the funds.

Here are the .h Files for all of the classes:

#include <queue>

#include "BSTree.h"

#include <string>

#include "Account.hpp"

#include "ApplyTransactions.h"

using namespace std;

class Bank

{

public:

Bank();

~Bank();

void ReadIn(string fileName); //file containing the transactions

void ExecuteTransactions(); // go through the queue and execute the transactions in order

void PrintOut(); // print all the open accounts and their balances

private:

queue<string> transactions; //contains the transactions

BSTree accountTree; // holds all accounts in the bank

};

#include "Funds.h"

#include <string>

#include <vector>

class Account

{

public:

Account(string first, string last, int id); // creates an account for the client

~Account();

string getFirstName() const;

string getLastName() const;

string getFullName() const; // get both the first and last name

int getAccID() const;

// setters in case anyone needs to change their name/id

bool setFirstName(string first);

bool setLastName(string last);

bool setID(int id);

void AddTransactionToHistory(string transactionType, int id1, int id2, int amount);

// vectors contain a string of the history for either all transactions or a single fund

vector<string> getAllHistory() const;

vector<string> getSingleFundHistory() const;

// checks if two accounts are equal to each other -> helps with not creating two of the same account

bool operator==(Account &other) const;

private:

string firstName;

string lastName;

int accID;

Funds myFunds[10]; // array of all the 10 funds each account can have

};

#include "CheckTransactions.h"

#include "Account.h"

class ApplyTransactions

{

public:

ApplyTransactions();

~ApplyTransactions();

// The five types of transactions

bool Open(string first, string last, int id);

bool Deposit(int id, int amount);

bool Withdraw(int id, int amount);

bool Transfer(int id1, int amount, int id2);

bool History(int id);

private:

bool HistoryOfAllTransactions(int id);

bool HisotryOfSingleFund(int id);

};

#include "Account.h"

class CheckTransactions

{

public:

CheckTransactions();

~CheckTransactions();

bool CanOpenAccount(int id, string first, string last); //makes sure the bank can open the account (no duplicates)

bool CanWithdraw(int id, int amount); //makes sure can withdraw (if can’t withdraw, then see if can take money from other funds)

bool CanTransfer(int id1, int amount, int id2); // makes sure there is enough balance to transfer and that both accounts are valid

bool CanDeposit(int id, int amount); // makes sure account trying to deposit to is a valid account

private:

};

#include <string>

#include <vector>

using namespace std;

class Funds

{

public:

Funds(string name);

~Funds();

void AddBalance(int amount, string transaction);

void RemoveBalance(int amount, string transaction);

int getBalance() const;

int getName() const;

vector<string> getFundHistory() const;

private:

string fundName;

int fundBalance;

vector<string> fundHistory; // keeps track of transactions for each fund

};

#include "Account.h"

class BSTree

{

public:

BSTree();

~BSTree();

bool Insert(Account \*);

bool Retrieve(const int &, Account \* &);

void Display() const;

void Empty();

bool isEmpty() const;

private:

struct Node

{

Account \*pAcct;

Node \*right;

Node \*left;

};

Node \*root;

};