**Christopher DuBois Lab 5 CSS 342 Dimpsey Winter Quarter 2015**

#include <queue>

#include <fstream>

#include <iostream>

using namespace std;

class Bank

{

public:

Bank();

bool ReadFile(string FileName);

bool ProcessTransactions();

bool OutputFile(string FileName) const;

bool Insert(Account\* that);

void Retrieve(const int& AccountID, Account\*& toRetrieve);

bool isEmpty() const;

bool Empty();

void Display() const;

Transaction getTransaction() const;

~Bank();

private:

queue<Transaction> Transactions;

BTree AccountHolder;

};

#include <string>

using namespace std;

class BTree

{

public:

BTree();

~BTree();

bool Insert(Account\* that);

// retrieve object, first parameter is the ID of the account

// second parameter holds pointer to found object, NULL if not found

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

// displays the contents of a tree to cout

void Display() const;

void Empty();

bool isEmpty() const;

private:

struct Node

{

Account\* data;

Node\* right;

Node\* left;

};

Node\* root;

Node\* Empty(Node\* current);

Node\* Insert(Account\* that, Node\* current, bool& insert);

void Retrieve(const int& AccountID, Account\*& toFind, Node\* current, bool& ret) const;

Node\* Display(Node\* current) const;

};

#pragma once

using namespace std;

#include <string>

#include <iostream>

class Transaction

{

friend istream& operator>>(istream& is, Transaction& right);

friend ostream& operator<<(ostream& os, const Transaction& right);

public:

Transaction();

Transaction(char action, int acctID, int fundid, int amount);

Transaction(char action, int acctID, int fundid, int amount, string failure);

Transaction(char action, int acctID, int fundid, int amount, int otherAcctID, int otherFundID);

Transaction(char action, int acctID, int fundid, int amount, string failure, int otherAcctID, int otherFundID);

char getTransactionAction() const;

int getTransactionAccountID() const;

int getTransactionFundID() const;

int getOtherTransactionAccountID() const;

int getOtherTransactionFundID() const;

string getFailure() const;

string getTransactionFirstName() const;

string getTransactionLastName() const;

string getOtherTransactionFirstName() const;

string getOtherTransactionLastName() const;

int getTransactionAmount() const;

~Transaction();

private:

char action;

int AccountID;

int FundID;

int otherAccountID;

int otherFundID;

string FirstName;

string LastName;

string otherFirstName;

string otherLastName;

string Failure;

int amount;

};

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Client

{

friend ostream& operator<<(ostream& os, const Client& right);

public:

Client();

Client(string LastName, string FirstName);

int getClientAccountID() const;

string getClientFirstName() const;

string getClientLastName() const;

~Client();

private:

string FirstName;

string LastName;

};

#pragma once

#include <vector>

#include "Fund.h"

#include "Client.h"

using namespace std;

class Account

{

friend ostream& operator<<(ostream& os, const Account& right);

public:

Account();

Account(int AccountID, Client AssociatedClient);

bool Deposit(const Transaction& Depositer);

bool Withdraw(const Transaction& Withdrawal);

bool Transfer(const Transaction& Transfer, Account& TransferTo);

void DisplayAccountTransactionsByFund() const;

void DisplaySpecificFundTransactionsInAccount(int TypeOfFund) const;

int getAccountID() const;

Client getClientAssociatedWithAccountID() const;

bool Exists() const;

Fund\* getFund(int index) const;

const bool operator<(const Account& toCompare) const;

const bool operator==(const Account& toCompare) const;

const bool operator>(const Account& toCompare) const;

void DeleteAccount();

~Account();

private:

int AccountID;

Client AssociatedClient;

vector <Fund\*> Funds;

};

#pragma once

#include <vector>

#include "Transaction.h"

#include <iostream>

#include <string>

using namespace std;

class Fund

{

friend ostream& operator<<(ostream& outStream, const Fund& toPrint);

public:

Fund();

Fund(string TypeOfFund);

bool DepositIntoFund(double amount);

bool WithdrawFromFund(double amount);

int getFundID() const;

string getFundType() const;

int getBalance() const;

void PrintTransactionHistory() const;

bool UpdateHistory(Transaction NewAction);

~Fund();

private:

int FundBalance;

int FundID;

string FundType;

vector<Transaction> History;

};