Ajay Vegiraju

**Jolly\_Banker Design**

*Design Explanation:*

1. Driver calls Bank.readIn() method which takes the opens a text file of transactions(name specified in command line) and converts them to Transaction Objects while populating a STL Queue with the Transaction Objects being read in.
   1. Each Transaction Object contains key information such as account numbers, fund numbers in which the transaction is taking place as well as the amount being transacted.
2. Driver calls Bank.Execute() method which accesses the queue of transactions objects in the Bank class. This queue is then popped one by one and the latest transaction that has been popped is processed
   1. O / open account transaction objects create and insert Account objects into a Binary Search Tree located in the Bank class.
   2. D/W/T transactions are meant to change the balance of funds in Account objects. Fund objects are stored in an array for each Account object are. Transactions access the Accounts in the BSTree using the retrieve() method located in the BSTree class.
   3. A/F display the account and fund histories respectively in the output file using an fstream object. Fund Histories are stored in a vector inside the fund class. Account histories are a collection of all of the fund histories of a specific Account object.
3. Driver calls Bank.Output() method which displays the final balances of all accounts in the BSTree using the help of the Display() method inside BSTree.
   1. Display method goes to each node in the BSTree, accesses the account object and then calls the displayFundBalances() method which is located in the Account class and displays all of the final fund balances for each Account. The balances are printed in the same file that the history transactions were outputted to.
4. All errors are printed to the console.

*Header Files:*

Jolly\_Bank.h:-

#ifndef JOLLY\_BANK\_H\_

#define JOLLY\_BANK\_H\_

#include "Transaction.h"

#include <queue>

#include <iostream>

#include "BSTree.h"

#include "Account.h"

#include <fstream>

#pragma once

using namespace std;

class Bank

{

public:

    Bank();

    ~Bank();

    //function to read in transactions from the file to the queue

    void ReadIn(const string file\_name);

    string Execute();

    void outPut();

private:

        queue<Transaction> transactions\_;

        BSTree accounts\_;

};

#endif

BSTree.h:-

#ifndef BSTREE\_H

#define BSTREE\_H

#include "Account.h"

#include <fstream>

#pragma once

class BSTree

{

public:

    BSTree();

    ~BSTree();

    bool Insert(Account\* account);

    // 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 &account\_id, Account \*&account) const;

    BSTree &operator=(BSTree &tree);

    bool isEmpty() const;

    int Size() const;

    // displays the contents of a tree to cout

    void Display() const;

private:

    struct Node

    {

        Node \*right = nullptr;

        Node \*left = nullptr;

        Account \*p\_acct;

        Node(Account \*account, Node \*left, Node \*right)

        {

            p\_acct = account;

            left = left;

            right = right;

        }

        Node(Account \*account){

            p\_acct = account;

            left = nullptr;

            right = nullptr;

        }

    };

    Node \*root\_ = nullptr;

    bool InsertHelper(Node \*cur, Account\* account);

    void MakeEmpty(Node\* root);

    void printHelper(Node \*root) const;

    int size\_;

};

#endif

Transaction.h:-

#ifndef TRANSACTION\_H\_

#define TRANSACTION\_H\_

#include <iostream>

#include <string>

#include <vector>

#pragma once

using namespace std;

class Transaction

{

public:

    Transaction();

    ~Transaction();

    Transaction(char type, string firstName, string lastName,int first\_account\_number\_);

    Transaction(char type, int accountNumber, int fundNumber, int amount);

    Transaction(char type, int accountNumber, int fundNumber, int amount, int transferAccountNumber, int transferFundNumber);

    Transaction(char type, int accountNumber);

    Transaction(char type, int accountNumber, int fundNumber);

    // Transaction(char trans\_type\_, string firstName, string lastName, int first\_account\_number\_);

    // Transaction(char type, int first\_account\_number = 0, int first\_acc\_fund = 0, int amount = 0, int sec\_account\_number = 0, int sec\_acc\_fund = 0);

    // void setName(string name);

    string GetFirstName();

    string GetLastName();

    int GetFirstAccountNumber() const;

    int GetSecondAccountNumber() const;

    int GetFirstAccFund() const;

    int GetSecondAccFund() const;

    int GetAmount() const;

    char GetTransType() const;

    string DisplayTransactions() const;

    void SetBool();

    friend ostream & operator<<(ostream &out, const Transaction &trans);

private:

    string first\_name\_;

    string last\_name\_;

    int first\_account\_number\_;

    int sec\_account\_number\_;

    int first\_acc\_fund\_;

    int sec\_acc\_fund\_;

    int amount\_;

    char trans\_type\_;

    bool passed\_trans\_ = true;

};

#endif

Account.h:-

#ifndef ACCOUNT\_H\_

#define ACCOUNT\_H\_

#include <string>

#include <iostream>

#include "Transaction.h"

#include "Fund.h"

#include <fstream>

using namespace std;

class Account

{

    public:

        Account();

        Account(int acc\_num, string first\_name, string last\_name);

        int getData() const;

        Account& operator=(const Account& other);

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

        bool operator!=(const Account& other) const;

        bool operator<(const Account& other) const;

        bool operator>(const Account& other) const;

        bool operator<=(const Account& other) const;

        bool operator>=(const Account& other) const;

        bool withdraw(Transaction transaction);

        bool deposit(Transaction transaction);

        bool transferInto(Transaction transaction);

        string displayFundBalances() const;

        string displayFundHistory(int fund\_number) const;

        string displayAccountHistory() const;

        int getAccNum();

        string getFirstName();

        string getLastName();

        friend ostream& operator<<(ostream& stream, const Account& other);

    private:

    int acc\_num\_;

    string first\_name\_;

    string last\_name\_;

    Fund funds[8];

};

#endif

Fund.h:-

#ifndef FUND\_H

#define FUND\_H

#include <string>

#include <vector>

#include "Transaction.h"

#pragma once

using namespace std;

class Fund

{

public:

    Fund();

    ~Fund();

    void AddAmount(int amount);

    void SubtractAmount(int amount);

    bool IsValid(int amount);

    void History( Transaction transaction);

    string DisplayHistory() const;

    void SetFundName(string name);

    int GetBalance() const;

    string GetFundName() const;

    friend ostream& operator<<(ostream& stream, const Fund& other);

    void SetBalanceToZero();

private:

    string fund\_name\_;

    int balance\_;

    vector<Transaction> fund\_history\_;

};

#endif

*A diagram of a flowchart

Description automatically generated with low confidenceDesign Visual:*

*Data Structure Visual:*

BSTree:  
Diagram

Description automatically generated