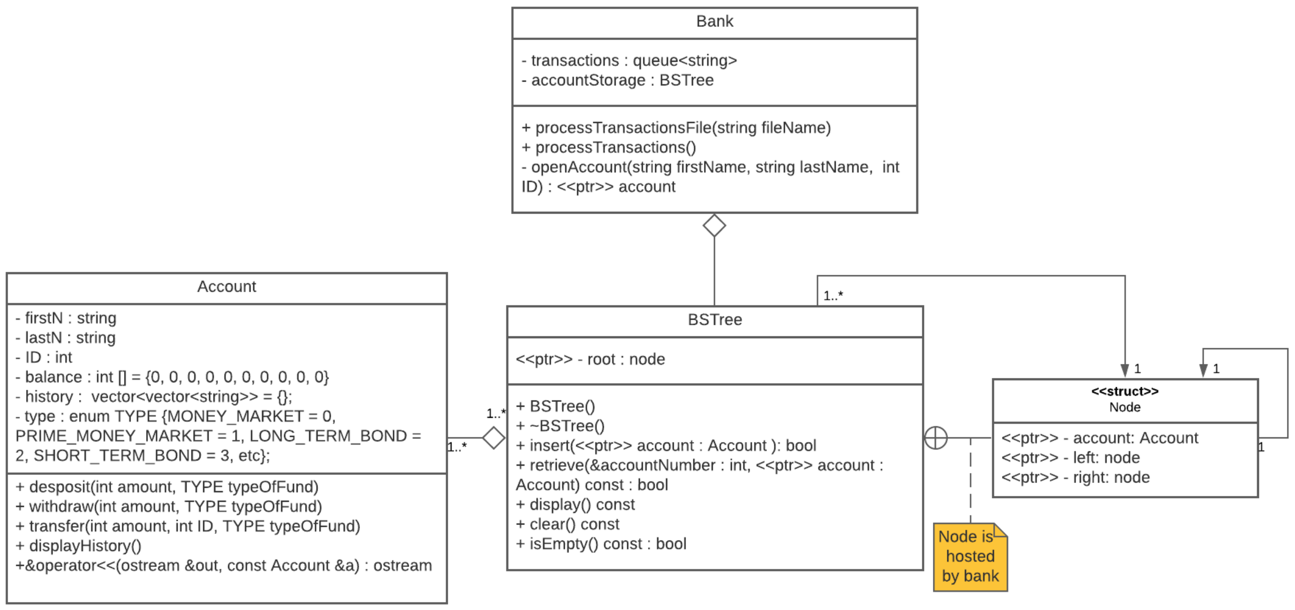
Project Design: Banking

by Reiner Optiz, Qiqi Hu, Trinh To, and Liam Morrison

Overview:

BankSoft is an application that processes transactions. Banksoft will read transactions from a file and place them in a queue. Banksoft will implement the transaction functionality to open accounts, withdraw funds, deposit funds, transfer, and print a transaction history.

Class Diagram and Descriptions


Bank
- transactions : queue<string>
- accountStorage : BSTree
+ processTransactionsFile(string fileName)
+ processTransactions()
- openAccount(string firstName, string lastName, int ID) : <<ptr>> account
Account
- firstN : string
- lastN : string
- ID : int
- balance : int [] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
- history : vector<vector<string>> = {};
- type : enum TYPE {MONEY_MARKET = 0, PRIME_MONEY_MARKET = 1, LONG_TERM_BOND = 2, SHORT_TERM_BOND = 3, etc};
+ desposit(int amount, TYPE typeOfFund)
+ withdraw(int amount, TYPE typeOfFund)
+ transfer(int amount, int ID, TYPE typeOfFund)
+ displayHistory()
+&operator<<(ostream &out, const Account &a) : ostream
1..*
1..*
BSTree
<<ptr>> - root : node
+ BSTree()
+ ~BSTree()
+ insert(<<ptr>> account : Account): bool
+ retrieve(&accountNumber : int, <<ptr>> account : Account) const : bool
+ display() const
+ clear() const
+ isEmpty() const : bool
1..*
1
1
1
<<struct>>
Node
<<ptr>> - account: Account
<<ptr>> - left: node
<<ptr>> - right: node
Node is hosted by bank

Bank

Description: Bank class will process and manage transactions. It receives input from a .txt file of transactions and stores the transactions as strings in a queue. It then processes each transaction that is in the queue.

```
#ifndef BANK_H
#define BANK_H

#include "bstree.h"
#include <string>
#include <queue>

using namespace std;

class Bank {
public:
  void processTransactionFile(const string &fileName);
  void processTransactions();

private:
  BSTree accounts;
  queue<string> transactions;
  Account *openAccount(string firstName, string lastName, int ID);
};
#endif // BANK_H
```

Account

Description: The Account class stores the first and last name of the account, the account ID, and the amount of money in each fund for the account, as well as a transaction history.

```cpp
#ifndef ACCOUNT_H
#define ACCOUNT_H

#include <iostream>
#include <vector>
#include <string>

using namespace std;

enum TYPE {
  MONEY_MARKET,
  PRIME_MONEY_BOND,
  LONG_TERM_BOND,
  SHORT_TERM_BOND,
  F_INDEX_FUND,
  CAPITAL_VALUE_FUND,
  GROWTH_EQUITY_FUND,
  GROWTY_INDEX_FUND,
  VALUE_FUND,
  VALUE_STOCK_INDEX
};

class Account {
  //print out a series of history
  friend ostream &operator << (ostream &out, const Account &other);
public:

 //constractor
 Account(string firstName, string lastName, int idNum);

 //operator to compare each account object by ID Number
 bool operator < (const Account &other) const;
 bool operator > (const Account &other) const;
 bool operator == (const Account &other) const;
 bool operator != (const Account &other) const;
 Account& operator = (const int &accountNum);

void deposit(int amount, TYPE typeOfFund);
 void withdraw(int amount, TYPE typeOfFund);
 void displayHistory();
 void transfer(int amount, Account *otherAccount, int typeOfFund1, int typeOfFund2);
 bool openFunds(int fundType);
 string getLastName();
 string getFirstName();
 int getIDNum();
 int getBalance(TYPE typeOfFund);

private:
  string firstName;
  string lastName;
  int idNum;
  int balance[10];
  vector<string> history[10];
  const string fund[10] = {
    "Money Market",
    "Prime Money Market",
    "Long-Term Bond",
    "Short-Term Bond",
    "500 Index Fund",
    "Capital Value Fund",
    "Growth Equity Fund",
    "Growth Index Fund",
    "Value Fund",
    "Value Stock Index"
  };
};
#endif // ACCOUNT_H
```

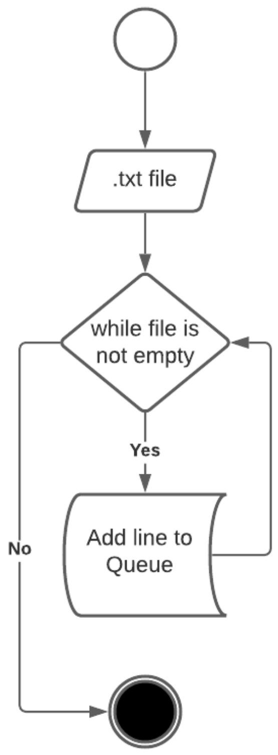
BSTree

Description: The BSTree class stores a tree of Accounts. Each Account within the tree is inserted or retrieved based on the Account ID.

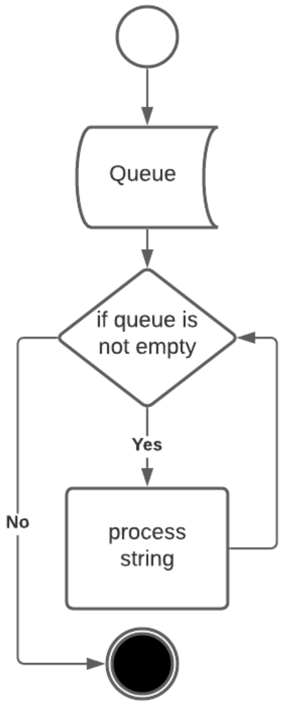
```
#ifndef BSTREE_H
#define BSTREE_H

#include "account.h"
#include <iostream>

class BSTree {
public:
  // Create BST
  BSTree();

  // Delete all nodes in BST
  ~BSTree();

  // Insert new account
  bool insert(Account *account);

  // Retrieve account
  // returns true if successful AND *Account points to account
  bool retrieve(const int &accountNumber, Account *&account) const;

  // Display information on all accounts
  void display() const;

  // delete all information in AccountTree
  void clear();

  // check if tree is empty
  bool isEmpty() const;

private:
  struct Node {
    Account *account;
    Node *right;
    Node *left;
  };
  Node *root;
};
#endif // BSTREE_H
```

Description of information flow for different cases

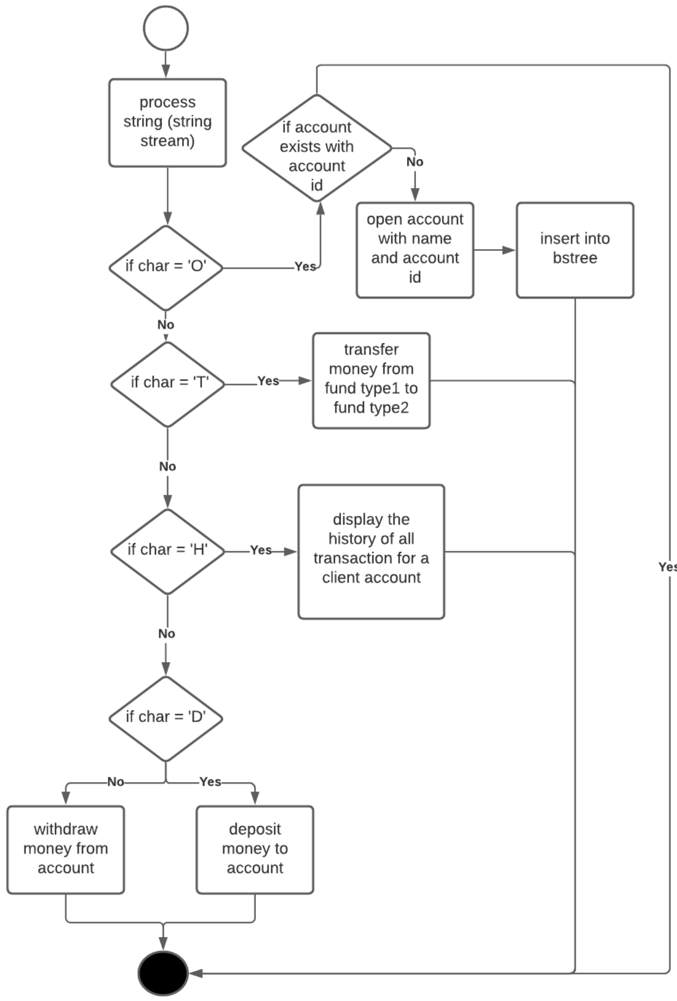
Case1: Start with a transaction text file input file line by line into queue.



Case 2: While the queue is not empty, process the queue in order.



Case 3: Process string transaction based on the character from the string.



process string (string stream)
if char = 'O'
Yes
if account exists with account id
No
open account with name and account id
insert into bstree
Yes
No
if char = 'T'
Yes
transfer money from fund type1 to fund type2
No
if char = 'H'
Yes
display the history of all transaction for a client account
No
if char = 'D'
No
Yes
withdraw money from account
deposit money to account

Read the file:

line by line - while can still getline(),

trim() and then put the line into our queue

→ a queue full of command

Process the queue:

Peek the string out of the queue->pop

Divide string into separate arguments

[0] Command type

```
If the type is ‘**O**’
    [1] First Name (string)
    [2] Last Name (string)
    [3] ID (int)
    _  Account(first, last, int) CONSTRUCTOR
    _  BSTree.insert( created account)
Else if the type is '**T**'
    [1] ID+Fundtype 1 (int)
    [2] Amount of money (int)
    [3] ID+Fundtype 2 (int)
    _ Account *a;
    _ Account *b;
    _ Int fundtype1 = [1] %10;
    _ Int fundtype2 = [3] % 10;
    _ BSTree.retrieve( [1] / 10, a)
    _ BSTree.retrieve( [3] / 10, b)
    _ a.transfer( [2] , b, fundtype1, fundtype2)
Else if the type is '**H**’
    [1] ID (int)
    _ Account *a;
    _ BStree.retrieve( [1], a)
    _ cout << *a;
Else
    [1] ID+Fund type (int)
    [2] Amount of money (int)
    _ Account *a;
    _ Int fundtype = [1] %10;
    _ BSTree.retrieve( [1] / 10, a)

    if the type is '**D**’
    _ a.deposit( [2], fundtype)

    Else the type is '**W**'
    _a.withdraw([2], fundtype)
```

High Level pseudocode included for important methods (flow-chart)

Pseudocode:

Example: (could stringstream into the queue)

processTransactionsInFile()

```
FILE file
   Open file
   IF (file is open)
        String line;
          line = file.LINE
          WHILE (line NOT empty)
                queue.ADD(line)
          END-WHILE
   END-IF
```

processTransactions()

```
WHILE (queue is not empty)
queue.POP(line)
Stringstream(line)
IF ('O')
          account.OPEN
IF ('D')
          account.DEPOSIT
IF ('W')
          account.WITHDRAWAL
IF('T')
          account.TRANSFER
IF('H')
          account.HISTORY
```

```
    END-WHILE
```

openAccount(firstName, lastName, ID)

```
    OPEN account (firstName, lastName, ID)
    IF(!bstree.RETRIEVE(ID, account))
        bestree.INSERT(account)
    END-IF
    IF the account ID we are opening already exists in the bstree then
     Cerr << ACCOUNT IS ALREADY OPEN, TRANSACTION REFUSED.<< endl;
     END-IF
```

depositFund(accountID, fund, int amount)

```
    IF(!account.RETRIEVE)
       Update history fund
       Cerr <<ACCOUNT DOES NOT EXIST CANNOT DEPOSIT << endl;
    ELSE
       Update balance of fund
       Update history fund
     END-IF;
```

withdrawalFund(accountID, fund, int amount)

```
    IF(!account.RETRIEVE)
    Cerr <<ACCOUNT DOES NOT EXIST CANNOT WITHDRAWAL << endl;
    Check balance of fund
    IF (fund(balance) < amount)
        IF (fund(balance) + associate fund(balance) < amount)
```

```
                Cerr << INSUFFICIENT FUNDS << endl;
                ELSE-
                INT remainder;
                fund(balance) +  associate fund(balance) - amount =  TEMP amount
                fund(balance) = 0;
                Associate fund(balance) = remainder;
                Update history fund;
                END-IF
        fund(balance) = fund(balance) - amount;
        Update history fund;
```

transferFund(accountFromID, int amount, accountToID)

```
        IF(!accountFromID.RETRIVE || !accountTo.ID)
        Cerr >> ACCOUNT NOT FOUND CANNOT PROCESS TRANSFER << endl;
        END-IF;
        IF(accountFromID.RETRIVE == accountToID.RETREIVE)
        accountFromID.RETRIVE;
        account.withdrawalFund;
        account.depositFund;
        END-IF;
        ELSE-
        accountFromID.RETRIEVE
        accountToID.RETRIEVE
        accountFromID.withdrawalFund
        Update history fund
```

```
    accountToID.depositFund
    Update history fund
    END-ELSE
```

accountHistory()

```
for i = 0 THROUGH history
    PRINT fundNames[i]
    PRINT funds[i]
    for j in history
        PRINT history[i][j]
```