John Neigel
Lab 05
CSS 342

Design Abstract Summary

General Purpose

The Jolly Banker system's purpose is to aid in management of computer-based client account records for a bank. The system will be comprised of multiple classes and data structures. The system's operations will be informed by an input file. While the input file is parsed, operations will be stored in a queue data structure with FIFO properties. After all operations are stored, the system will begin conducting the requested operations. Operations will include opening new accounts, depositing assets into funds, withdrawing from funds, transfers between funds, and displaying history of a specified account.

Class and ADT Design

All client data will be stored within nodes of a binary search tree (BST). Each client will also contain its own BST of accounts. Both BSTs will be ordered by their client ID numbers, and account numbers, respectively.

The account class will be a superclass, passing through inheritance some common data fields shared by a set of concrete account types. Each account contains data fields of an account ID and the amount of funds currently in the account. Operations conducted by the account will facilitate the needed functional requirements above. Special note should be given to the withdrawal operation. This operation will be overridden, as it will need separate implementations for different accounts.

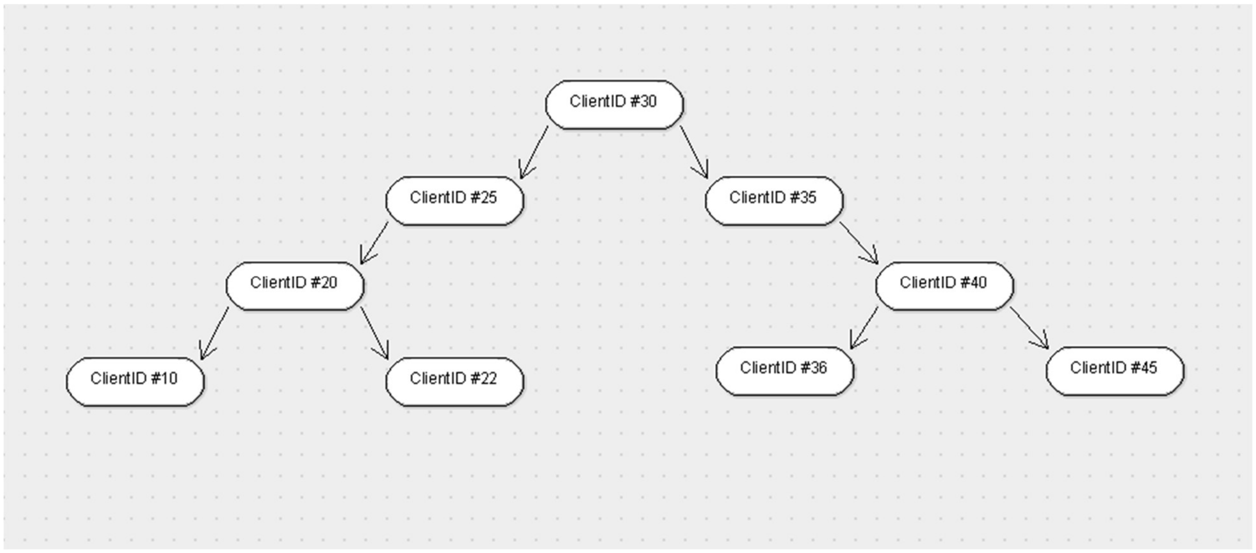
Errors Cases

Since it is assumed the input file is well informed, improper input does not need to be handled. Other errors, which will need to be addressed include requesting operations on a non-existent account, attempts to create a new account which already exists, attempts to withdraw with inadequate funds, attempts to transfer with inadequate funds. There may be others, but these must be specifically handled.

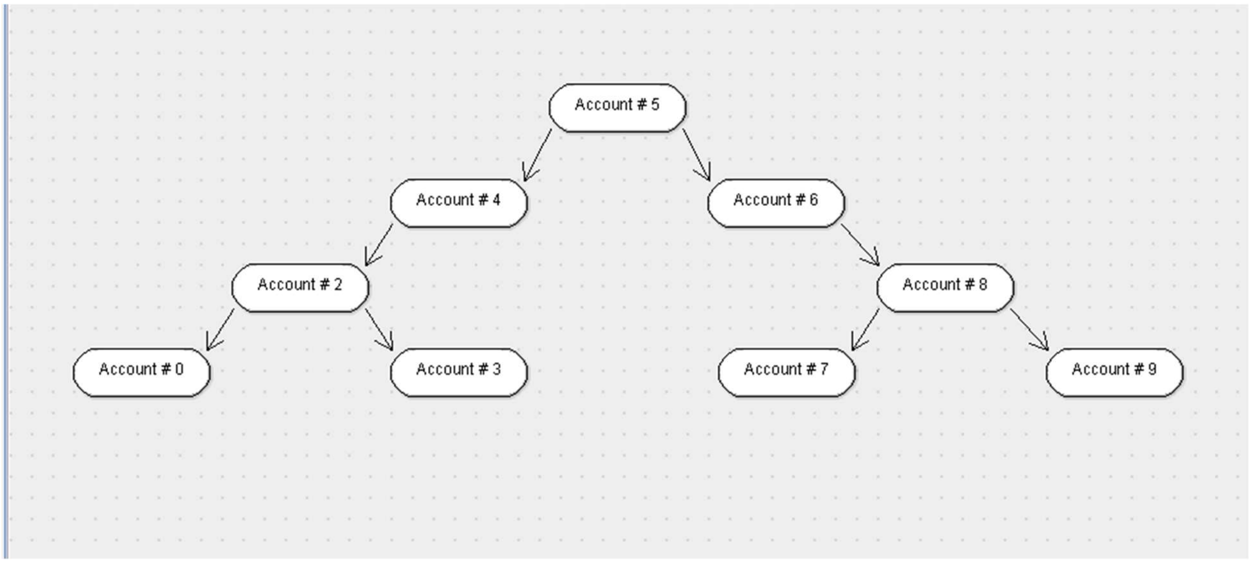
John Neigel
Lab 05
CSS 342

Design Abstract Summary

Client Tree



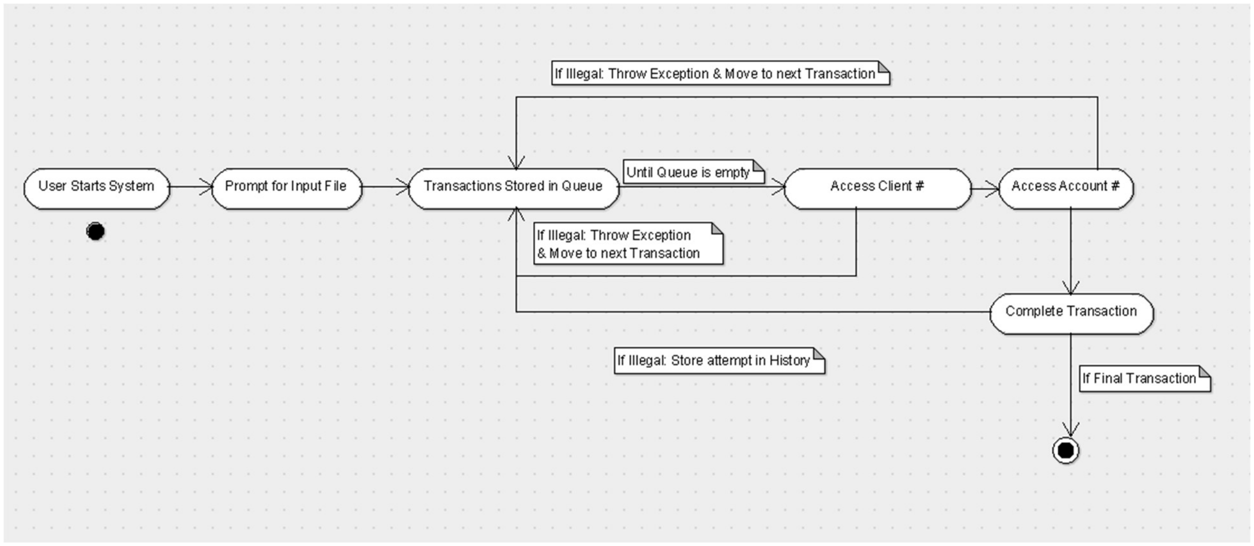
Account Tree



John Neigel
Lab 05
CSS 342

Design Abstract Summary

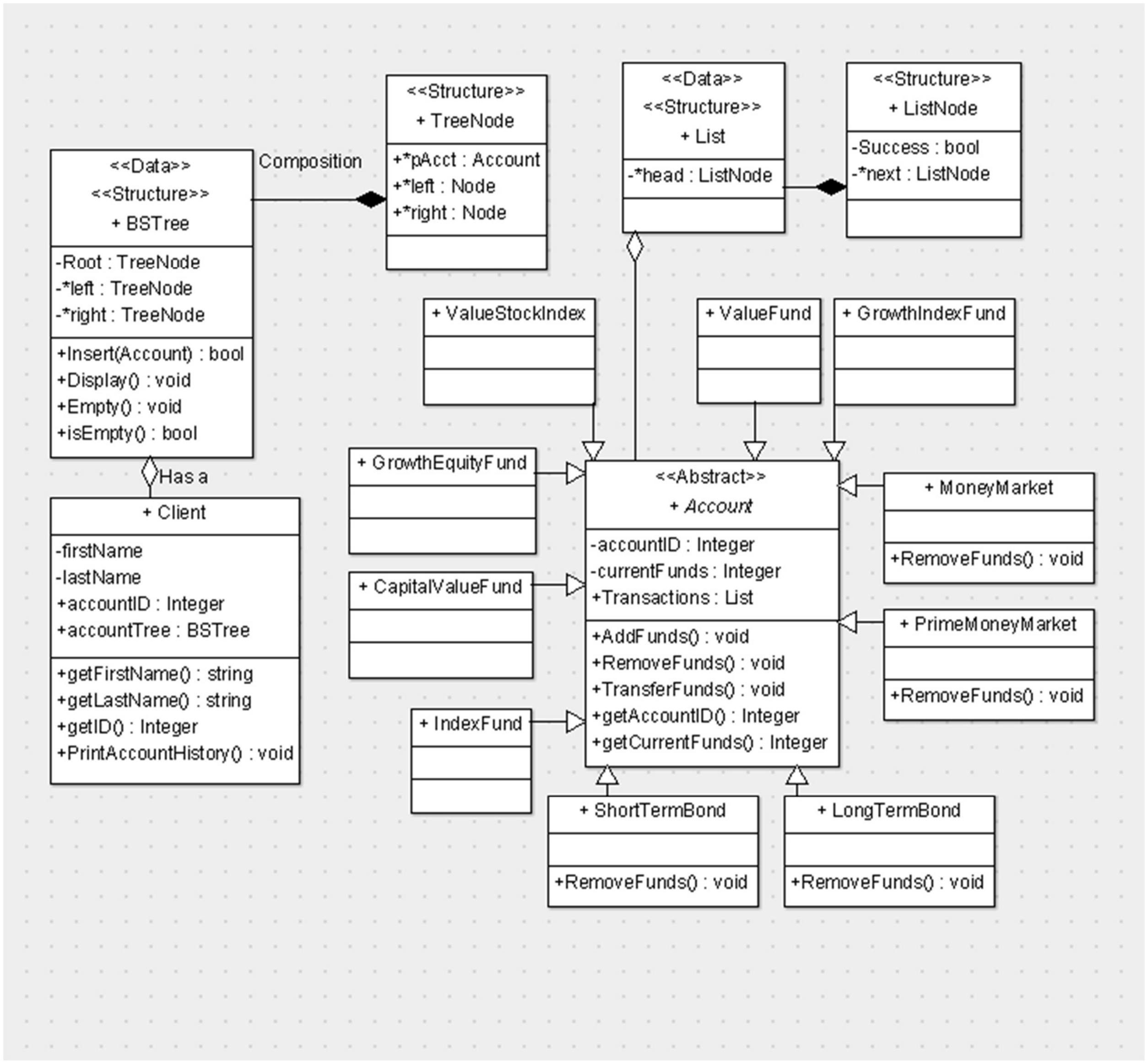
Sequence Diagram


If Illegal: Throw Exception & Move to next Transaction
User Starts System
Prompt for Input File
Transactions Stored in Queue
Until Queue is empty
Access Client #
Access Account #
If Illegal: Throw Exception
& Move to next Transaction
Complete Transaction
If Illegal: Store attempt in History
If Final Transaction

John Neigel
Lab 05
CSS 342

Design Abstract Summary

Class UML Design



JollyBanker.h

```
/*Purpose: This file contains the interface for the Jolly Banker system.
Contained are the interfaces for ADTs and classes utilized in the system.
Author: John Neigel
Last Edit: 05/30/2019
*/

#pragma once

//ADT for Binary Search Tree
```

John Neigel
Lab 05
CSS 342

Design Abstract Summary

```
//Templatized for use with client tree and account tree
template<class T> class BSTree {

public:

	//constructors & destructors
	BSTree();
	~BSTree();

	//functions
	bool Insert(T* objectPtr);
	bool Retrieve(const int& num, T*& objectPtr) const;
	void Display() const;
	void Empty();
	bool isEmpty() const;

private:

	//struct to comprise BSTree
	struct TreeNode
	{
		T* objectPtr;
		TreeNode* left;
		TreeNode* right;
	};

	TreeNode* root;
};

//templatized Queue implemented as a linked list
//used in storing transactions from file and storing account history
template<class T> class Queue
{
public:

	void push(T* objectPtr);
	T& pop();
	bool isEmpty();

private:

	struct QueueNode
	{
		T* objectPtr;
		T* next;
	};

	QueueNode* head;
};

class Transaction
{
public:

	Transaction();
```

John Neigel
Lab 05
CSS 342

Design Abstract Summary

```
        ~Transaction();

private:

        int targetAccount;
        bool successful;
        char type;
};

//abstract super class for client accounts
//all data fields are public for inheritance
class Account
{

public:
        int getAccountID();
        int getCurrentFunds;
        void AddFunds();
        void RemoveFunds();
        void TransferFunds();

private:

        int accountID;
        int currentFunds;
        Queue<Transaction> history;

};

class MoneyMarket : public Account
{
public:

        MoneyMarket();
        ~MoneyMarket();
        void RemoveFunds(); //override to handle withdraw differently
};

class PrimeMoneyMarket : public Account
{
public:

        PrimeMoneyMarket();
        ~PrimeMoneyMarket();
        void RemoveFunds(); //override to handle withdraw differently
};

class LongTermBond : public Account
{
public:

        LongTermBond();
        ~LongTermBond();
        void RemoveFunds(); //override to handle withdraw differently
};
```

John Neigel
Lab 05
CSS 342

Design Abstract Summary

```
class ShortTermBond : public Account
{
public:

	ShortTermBond();
	~ShortTermBond();
	void RemoveFunds(); //override to handle withdraw differently
};

class IndexFund : public Account
{
public:

	IndexFund();
	~IndexFund();
};

class CapitalValueFund : public Account
{
public:

	CapitalValueFund();
	~CapitalValueFund();
};

class GrowthEquityFund : public Account
{
public:

	GrowthEquityFund();
	~GrowthEquityFund();
};

class GrowthIndexFund : public Account
{
public:

	GrowthIndexFund();
	~GrowthIndexFund();
};

class ValueFund : public Account
{
public:

	ValueFund();
	~ValueFund();
};

class ValueStockIndex : public Account
{
public:

	ValueStockIndex();
```

John Neigel
Lab 05
CSS 342

Design Abstract Summary

```
	~ValueStockIndex();
};
```