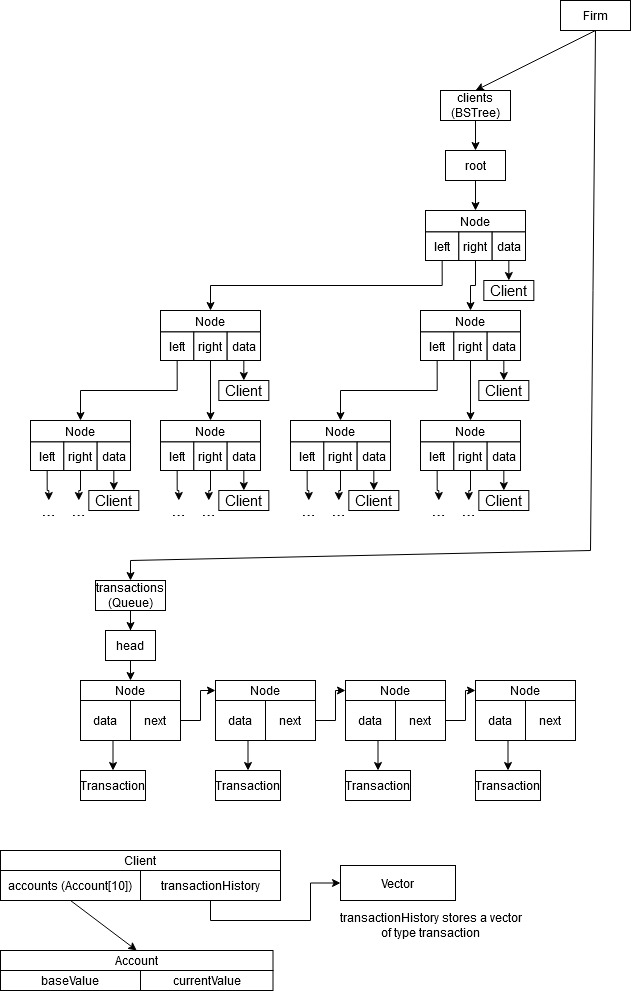
# UML Diagram

# 

# Data Structure Diagram



# Use Cases

firm::addClients(infile){

while still lines in file {

Declare Client object pointer // Client ptr;

Read last name, first name, accountID, and ten account values from infile

Break file line into each parameter

Initialize client to account information

Insert Client into client Binary Tree

}

}

//Case 1: D 12341 100

//Bolded code is executed in this case

**firm::addTransactions(infile){**

**while file is not empty {**

**Declare Transaction object;**

**if(the first char in the line from infile is ‘D’) {**

**Use infile to read in Transaction object data members;**

**// amount, firstAccount**

**Enqueue into queue**

**}**

Else if(the first char in the line from infile is ‘W’) {

Use infile to read in Transaction object data members;

// amount, firstAccount

Enqueue into queue

}

Else if(the first char in the line from infile is ‘M’) {

Use infile to read in Transaction object data members;

// amount, firstAccount, secondAccount

Enqueue into queue

}

Else if (the first char in the line from infile is ‘H’) {

Use infile to read in Transaction object data members;

//firstAccount

Enqueue into queue

}

Else {

Error message

}

}

}

**firm::processTransactions(){**

**while transaction remaining in transactions queue {**

**Dequeue transaction from queue**

**if(transaction command is ‘D’) {**

**Retrieve client from binary search tree using clientID**

**Perform deposit on Client depending on account number**

**Add transaction to client history**

**// find account from account array in client object**

**}**

Else if(transaction command is ‘W’) {

Retrieve client from binary search tree using clientID

Check if valid withdrawal

Perform withdrawal on Client

Add transaction to client history

}

Else if(transaction command is ‘M’) {

Check if valid withdrawal on first client

Find first client using getFrom()

Find second client using getTo()

Find amount of money to move // getAmount();

Add transaction to clients history

}

Else if(transaction command is ‘H’) {

Retrieve client from binary search tree using clientID

Show Client history

}

}

}

//Case 2: M 12340 1000 12341

//Bolded code is executed in this case

**firm::addTransactions(infile){**

**while file is not empty {**

**Declare Transaction object;**

if(the first char in the line from infile is ‘D’) {

Use infile to read in Transaction object data members;

// amount, firstAccount

Enqueue into queue

}

Else if(the first char in the line from infile is ‘W’) {

Use infile to read in Transaction object data members;

// amount, firstAccount

Enqueue into queue

}

**Else if(the first char in the line from infile is ‘M’) {**

**Use infile to read in Transaction object data members;**

**// amount, firstAccount, secondAccount**

**Enqueue into queue**

**}**

Else if (the first char in the line from infile is ‘H’) {

Use infile to read in Transaction object data members;

//firstAccount

Enqueue into queue

}

Else {

Error message

}

}

}

**firm::processTransactions(){**

**while transaction remaining in transactions queue {**

**Dequeue transaction from queue**

if(transaction command is ‘D’) {

Retrieve client from binary search tree using clientID

Perform deposit on Client depending on account number

Add transaction to client history

// find account from account array in client object

}

Else if(transaction command is ‘W’) {

Retrieve client from binary search tree using clientID

Check if valid withdrawal

Perform withdrawal on Client

Add transaction to client history

}

**Else if(transaction command is ‘M’) {**

**Check if valid withdrawal on first client**

**Find first client using getFrom()**

**Find second client using getTo()**

**Find amount of money to move // getAmount();**

**Add transaction to clients history**

**}**

Else if(transaction command is ‘H’) {

Retrieve client from binary search tree using clientID

Show Client history

}

}

}