Group: 7
Members: Jesse Luo and Tyler Wong
CSS 343 Assignment 4 Design
March 3rd, 2016

Overview

The main creates an instance of the Store object. A Store has 3 data members, a VideoManager, CustomerManager, and TransactionManager. The VideoManager acts as an inventory that holds 3 trees, one for each kind of video, and uses the VideoFactory to create various types of videos to be put into those trees. The CustomerManager creates Customers and puts them into a HashTable. The key will be the customer's ID and the value will be the Customer. Each Customer also has a Queue of Transactions which holds an ordered Linked List of all the Transactions this Customer has performed. The TransactionManager uses the TransactionFactory to create various types of Transactions which will be inserted into the corresponding customer's transaction log. The TransactionManager also holds a Queue of Transactions that acts as a history of the transactions performed in the Store. "Borrow" and "Return" transaction objects also contain a pointer to the Video that they are associated with.

Description of Main:
Create new Store object.
Call the Store's function buildInventory from data4movies.txt
Call the Store's function buildCustomerList from data4customers.txt
Call the Store's function performTransactions from data4commands.txt

Problem Description

A video rental store wants to automate their inventory tracking system. They want to manage all their videos, customers, and transactions so that they can track them. They want videos to be organized into genre groups. Customers should be easily stored and easily found by their ID. Transactions should be sorted into types and associated with a Customer so that they can have a transaction history.

Assumptions and Design Considerations

Assumptions

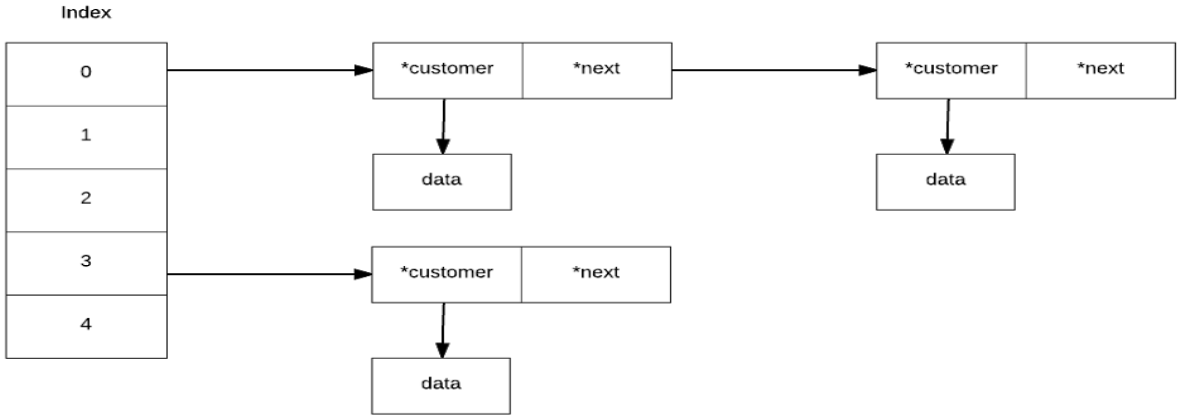
- There is no incorrectly placed commas in the input files.
- The input files are formatted correctly.
- Duplicate customer IDs are not allowed.

Design Considerations

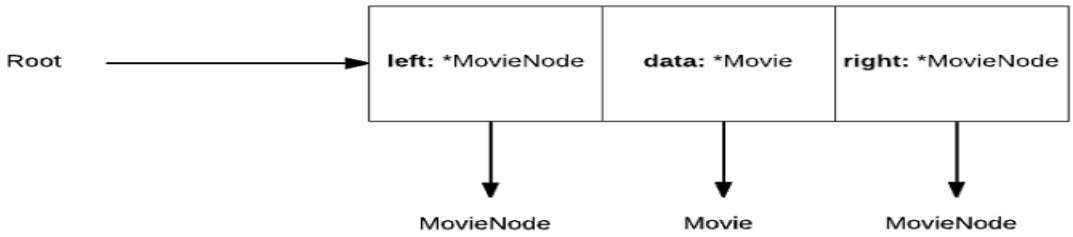
- We're using a HashTable for the list of customers because they already have a customer ID from which we can compute a hash.
- Factories are being used so that an interface can create an object, but also allowing the subclasses decide which class to instantiate.
- We're using three trees to store the inventory because they need to be sorted in a certain order.
- We're using a vector to store the history of transactions because it allows for fast gets of the data that we need without a lot of memory overhead.
- Each Customer will have a vector pointers to Transactions. The pointers will point to the transaction associated with that customer in the vector that holds all the transactions.

High-Level Algorithms

We are using a HashTable to keep track of all customers that the store currently has. Also, this allows for easy and fast access to any customer. This implementation will reduce searching time for a given customer.



In addition, we are using BSTrees in the InventoryManager to store information of all the movies that it currently has. There will be 1 BSTree for each genre of movie and will be sorted based off of the specification. This implementation will reduce searching time and allow for all the movies to be sorted on different variables.



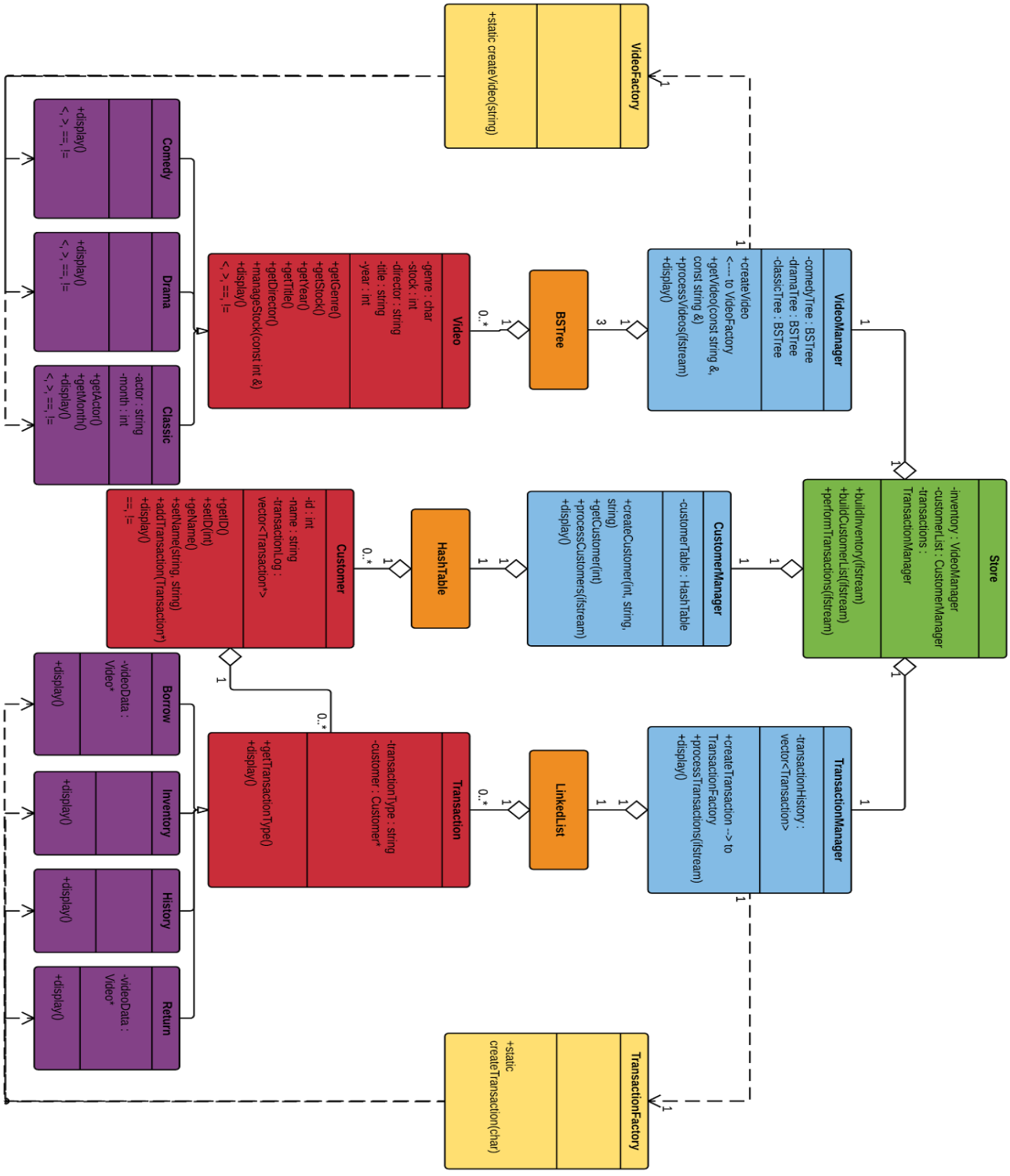
```
VideoManager::processVideos(ifstream)
{
      read in file
      for each video command
            call VideoFactory::createVideo(video)
            create new comedy, drama, or classic
            place into appropriate BSTree
}

CustomerManager::processCustomers(ifstream)
{
      read in file
      for each customer
            create new customer
            add to HashTable
}

TransactionManager::processTransactions(ifstream)
{
      read in file
      for each transaction command
            call TransactionFactory::createTransaction(transaction)
            create new borrow, inventory, history, or return
            place into queue
}

VideoManager::display()
{
      for each BSTree
            traverse BSTree in-order
            call display() on movie
}
```

Class Diagram



Class Descriptions (in .h files)

```
#include "VideoManager.h"
#include "CustomerManager.h"
#include "TransactionManager.h"
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

class Store
{

public:
	Store();
	~Store();

	void buildInventory(ifstream &infile);
	void buildCustomerList(ifstream &infile);
	void performTransactions(ifstream &infile);

private:
	VideoManager inventory;
	CustomerManager customerList;
	TransactionManager transactions;
};

#include "BinTree.h"
#include "VideoFactory.h"
#include "Video.h"
```

```
class VideoManager
{

public:
	VideoManager();
	~VideoManager();

	// return pointer to video in tree, find video by title (or actor if classical)
	Video* getVideo(const string &title, const string &actor);

	void createVideo(string newVideo);
	void processVideos(ifstream &infile);

	void display();

private:
	BinTree comedyTree;
	BinTree dramaTree;
	BinTree classicTree;
};

class VideoFactory
{

public:
	static Video* createVideo(string newVideo);
};
```

```
#include "Video.h"
#include <iostream>

class BinTree
{

public:
	BinTree();
	~BinTree();

	bool isEmpty() const;
	void makeEmpty();

	bool retrieve(const string &title, const string &actor, Video* &data);

	bool insert(Video* data); // No duplicate videos added, will increment stock instead

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

	BinTree& operator=(const BinTree &other);

private:
	struct Node
	{
		Video* data;
		Node* left;
		Node* right;
	};
	Node* root;

	// Private "helper" recusrive functions that pair with public
	// functions to recurisvely iterate over the tree.

	//used with makeEmpty() function
	void clear(Node* &current);

	// used with retrieve() function
	Video* retrieveNode(Node* current, const string &title, const string &actor);

	// used with insert() function
	void insertNode(Node* &current, Video* data);
};
```

```
#include <iostream>
#include <string>

using namespace std;

class Video
{

public:
	Video();
	~Video();

	void manageStock(int newStock);

	char getGenre() const;
	int getStock() const;
	int getYear() const;
	string getTitle() const;
	string getDirector() const;

	virtual void display();

	virtual bool operator==(const Video &other) const = 0;
	virtual bool operator!=(const Video &other) const = 0;
	virtual bool operator<(const Video &other) const = 0;
	virtual bool operator>(const Video &other) const = 0;

protected:
	char genre;
	int stock;
	string director;
	string title;
	int year;
};
```

```
#include "Video.h"

class Classic : public Video
{
public:
	Classic(string newVideo);
	~Classic();

	string getActor() const;
	int getMonth() const;

	void display();

	virtual bool operator==(const Video &other) const;
	virtual bool operator!=(const Video &other) const;
	virtual bool operator<(const Video &other) const;
	virtual bool operator>(const Video &other) const;

protected:
	string actor;
	int month;
};

#include "Video.h"

class Comedy : public Video
{
public:
	Comedy(string newVideo);
	~Comedy();

	void display();

	virtual bool operator==(const Video &other) const;
	virtual bool operator!=(const Video &other) const;
	virtual bool operator<(const Video &other) const;
	virtual bool operator>(const Video &other) const;
};
```

```
#include "Video.h"

class Drama : public Video
{
public:
	Drama(string newVideo);
	~Drama();

	void display();

	virtual bool operator==(const Video &other) const;
	virtual bool operator!=(const Video &other) const;
	virtual bool operator<(const Video &other) const;
	virtual bool operator>(const Video &other) const;
};

#include "HashTable.h"
#include "Customer.h"

class CustomerManager
{

public:
	CustomerManager();
	~CustomerManager();

	void createCustomer();
	Customer* getCustomer(int id);

	void processCustomers(ifstream &infile);
	void display();

private:
	HashTable customerList;
};
```

```
#include "Customer.h"

const int TABLESIZE = 1000;

class HashTable
{

public:
	HashTable();
	~HashTable();

	void put(Customer* newCustomer, int id);
	void get(int id);

	void display();

private:
	struct Node
	{
		Node* next;
		Customer* data;
	};

	Node* customertable[TABLESIZE]; // intialize to NULL in constructor
};
```

```
#include "Transaction.h"
#include <iostream>
#include <string>

using namespace std;

class Customer
{

public:
	Customer(string newCustomer);
	~Customer();

	int getID() const;
	string geName() const;

	void setID(int id);
	void setName(string name);
	void addTransaction(Transaction* newTransaction);

	void display();

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

private:
	vector<Transaction*> transactionLog;
	int id;
	string name;
};
```

```
#include "Transaction.h"
#include "TransactionFactory.h"
#include "CustomerManager.h"
#include "VideoManager.h"
#include <vector>

class TransactionManager
{

public:
	TransactionManager();
	~TransactionManager();

	void createTransaction(string newTransaction);

	void processTransactions(ifstream &infile);
	void display();

private:
	vector<Transaction> transactionHistory;
};

class TransactionFactory
{

public:
	static Transaction* createTransaction(string newTransaction);
};
```

```
#include <string>

using namespace std;

class Transaction
{

public:
	Transaction();
	~Transaction();

	virtual string getTransactionType() = 0;
	virtual void display() = 0;

protected:
	string transactionType;
	Customer* customer;
};

#include "Video.h"
#include "Transaction.h"

using namespace std;

class Borrow : public Transaction
{

public:
	Borrow();
	~Borrow();

	virtual void display();

protected:
	Video* videoData;
};
```

```
#include "Video.h"
#include "Transaction.h"

using namespace std;

class Return : public Transaction
{

public:
	Return();
	~Return();

	virtual void display();

protected:
	Video* videoData;
};

#include "Transaction.h"
#include "CustomerManager.h"

using namespace std;

class History : public Transaction
{

public:
	History();
	~History();

	virtual void display();
};
```

```
#include "Transaction.h"
#include "VideoManager.h"

using namespace std;

class Inventory : public Transaction
{

public:
	Inventory();
	~Inventory();

	virtual void display();
};
```