Adam Ali, Omar Aguirre, Hyosang Park, Adam Mirza
CSS 343 A
Program 4 Design
May 23, 2017

Overview:

This design describes a movie store checkout system. Several different classes describe the major components of this checkout system, which makes it modular. The design is also extensible by means of inheritance to easily add more functionality or types of media (be it other media types or simply more categories of movies) .

Of the several classes, there are three major objects that the design involves:

Customer, Media, and Transaction.

These classes describe components of the movie store checkout system that would be included in collections.

Customer keeps track of identification attributes and transaction history.

Media and its derivatives (for now, just *Movie – which has genre subclasses*) record stock counts and metadata (titles, names, dates, etc.).

Transaction encapsulates any operation that a given customer might execute, including the transaction type and the *Media* item associated with it.

Additionally, this design includes a binary tree class *BinTree* to store *Media* objects for efficient retrieval. This leaves the *Store* class, which includes the *Customer* hash table, the *Media* inventory, file reading operations, transaction execution, display operations.

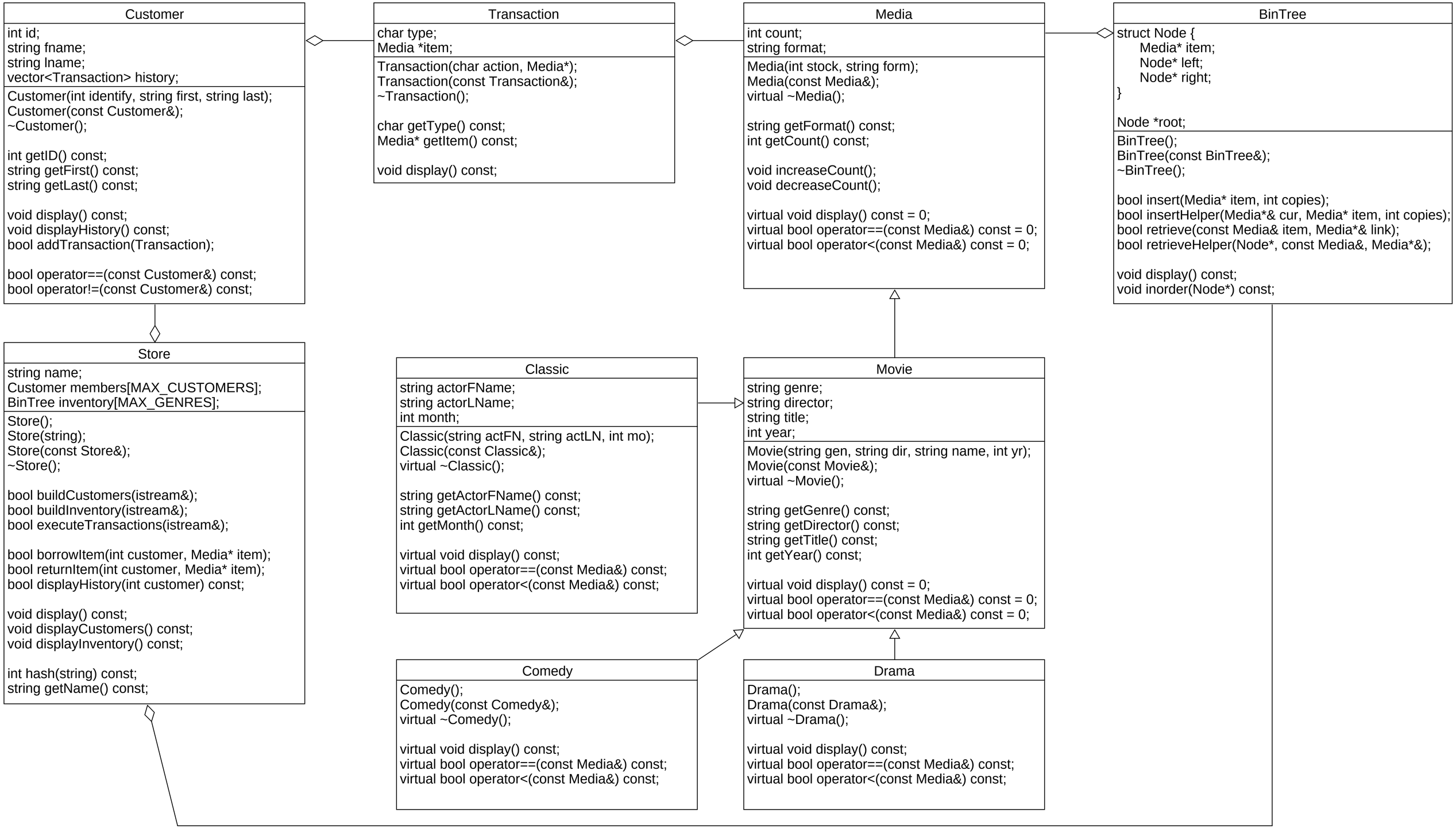
Beyond main, the program is driven through the *Store* object representing one given store. This means that a chain of stores can be easily implemented from a client.

Description of main:

Main is only responsible for creating *ifstream* objects for the three external data files, and constructing a *Store* object.

Objects found in main:

ifstream	*x 3*
Store	*x 1*


Customer
int id;
string fname;
string lname;
vector<Transaction> history;
Customer(int identify, string first, string last);
Customer(const Customer&);
~Customer();
int getID() const;
string getFirst() const;
string getLast() const;
void display() const;
void displayHistory() const;
bool addTransaction(Transaction);
bool operator==(const Customer&) const;
bool operator!=(const Customer&) const;
Transaction
char type;
Media *item;
Transaction(char action, Media*);
Transaction(const Transaction&);
~Transaction();
char getType() const;
Media* getItem() const;
void display() const;
Media
int count;
string format;
Media(int stock, string form);
Media(const Media&);
virtual ~Media();
string getFormat() const;
int getCount() const;
void increaseCount();
void decreaseCount();
virtual void display() const = 0;
virtual bool operator==(const Media&) const = 0;
virtual bool operator<(const Media&) const = 0;
BinTree
struct Node {
Media* item;
Node* left;
Node* right;
}
Node *root;
BinTree();
BinTree(const BinTree&);
~BinTree();
bool insert(Media* item, int copies);
bool insertHelper(Media*& cur, Media* item, int copies);
bool retrieve(const Media& item, Media*& link);
bool retrieveHelper(Node*, const Media&, Media*&);
void display() const;
void inorder(Node*) const;
Store
string name;
Customer members[MAX_CUSTOMERS];
BinTree inventory[MAX_GENRES];
Store();
Store(string);
Store(const Store&);
~Store();
bool buildCustomers(istream&);
bool buildInventory(istream&);
bool executeTransactions(istream&);
bool borrowItem(int customer, Media* item);
bool returnItem(int customer, Media* item);
bool displayHistory(int customer) const;
void display() const;
void displayCustomers() const;
void displayInventory() const;
int hash(string) const;
string getName() const;
Classic
string actorFName;
string actorLName;
int month;
Classic(string actFN, string actLN, int mo);
Classic(const Classic&);
virtual ~Classic();
string getActorFName() const;
string getActorLName() const;
int getMonth() const;
virtual void display() const;
virtual bool operator==(const Media&) const;
virtual bool operator<(const Media&) const;
Movie
string genre;
string director;
string title;
int year;
Movie(string gen, string dir, string name, int yr);
Movie(const Movie&);
virtual ~Movie();
string getGenre() const;
string getDirector() const;
string getTitle() const;
int getYear() const;
virtual void display() const = 0;
virtual bool operator==(const Media&) const = 0;
virtual bool operator<(const Media&) const = 0;
Comedy
Comedy();
Comedy(const Comedy&);
virtual ~Comedy();
virtual void display() const;
virtual bool operator==(const Media&) const;
virtual bool operator<(const Media&) const;
Drama
Drama();
Drama(const Drama&);
virtual ~Drama();
virtual void display() const;
virtual bool operator==(const Media&) const;
virtual bool operator<(const Media&) const;

```
// ---------------------------------store.h---------------------------------
// Adam Ali CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// -------------------------------------------------------------------------
// Describes the ADT Store such that collections of information + records
// can be maintained to represent a movie store that is setup for media item
// checkouts, much like a library. Any given Store will maintain a record
// of media items, customers, and the transaction history of said customers.
// -------------------------------------------------------------------------
// Functionality includes:
//              - create a no-name Store
//              - create a named Store
//              - copy an existing Store
//              - destruct a Store
//              - build a Customer hash table from file
//              - build a Media tree hash table from file
//              - honor borrow/return/history requests for any Customer
//              - output all Media and Customers of a Store

#pragma once

#include <string>
#include <iostream>
#include "customer.h"
#include "media.h"
#include "bintree.h"
using namespace std;

class Store {
        public:
                // -------------------------------Store-----------------------------
                // Store: creates a no-name Store.
                // preconditions: none.
                // postconditions: a no-name Store is created.
                // -----------------------------------------------------------------
                Store();


                // -------------------------------Store-----------------------------
                // Store: creates a named Store.
                // preconditions: none.
                // postconditions: a named Store is created.
                // -----------------------------------------------------------------
                Store(std::string);


                // -------------------------------Store-----------------------------
                // Store: creates a copy of the other Store.
                // preconditions: none.
                // postconditions: a copy Store is created.
                // -----------------------------------------------------------------
                Store(const Store&);


                // -------------------------------~Store----------------------------
                // ~Store: destructs the Store and frees any assoc. memory.
                // preconditions: none.
                // postconditions: any assoc. memory is freed, object inaccessible.
                // -----------------------------------------------------------------
                ~Store();


                // ---------------------------buildCustomers-------------------------
```

```
// buildCustomers: builds Customer hash table from file.
// preconditions: file is properly formatted, i.e. each line is
//                                #### fname lname
//                                given 4 digit IDs, the maximum allowed is 10,000.
// postconditions: customer hash table populated with all entries.
// -----------------------------------------------------------------
bool buildCustomers(istream&) {
        // while lines are available
                // read current line
                // extract id, fname, lname
                // build a Customer with extracted data
                // generate hash from id
                // set members[hash % MAX_CUSTOMERS] = freshly built Customer
}


// ---------------------------buildInventory-------------------------
// buildInventory: builds Media tree from file.
// preconditions: file is properly formatted, i.e. each line is
//                                X, ##, dfname dlname, title, . . .
//                                genre, stock, director name, title, and more
attribs
//                                comma delimited
// postconditions: BinTree populated with media items.
// -----------------------------------------------------------------
bool buildInventory(istream&) {
        // while lines are available
                // read current line
                // extract genre, stock, director, title
                // if genre is Drama/Funny
                        // extract year
                        // build Drama/Funny
                // else
                        // extract actor fname/lname, month, year
                        // build Classic
                // hash genre char
                // inventory[hash % MAX_GENRES].insert(Media)
}


// -------------------------executeTransactions---------------------
// executeTransactions: process transactions from file and update
//                                                 records as needed.
// preconditions: file is properly formatted, i.e. each line is one of
//                                X #### X X # #### fname lname
//                                X #### X X Title, ####
//                                X #### X X fname lname, title
//                                X ####
//                                X
// postconditions: records updated according to transactions.
// -----------------------------------------------------------------
bool executeTransactions(istream&) {
        // while lines available
                // read current line
                // extract trans type
                // switch ( trans type )

                        // case I : inventory.display()
                        // case H : read id
                                                // displayHistory(id);
                        // case B : read id, format, genre
                                                // if Funny, read title, year
                                                // if Drama, read director, title
                                                // if Classic, read month, year,
actor f/lname
```

```
                                                        // build Movie item from collected
  data
                                                        // inventory.retrieve(Movie, ptr);
                                                        // borrowItem(id, ptr);
                                        // case R : read id, format, genre
                                                        // if Funny, read title, year
                                                        // if Drama, read director, title
                                                        // if Classic, read month, year,
  actor f/lname
                                                        // build Movie item from collected
  data
                                                        // inventory.retrieve(Movie, ptr);
                                                        // returnItem(id, ptr);
                }


                // ------------------------------borrowItem---------------------------
                // borrowItem: applies a borrow transaction for a particular customer
                //                          + media item.
                // preconditions: customer exists, 0 < id < 10,000
                //                             media item exists.
                // postconditions: customer transaction history + item stock updated.
                // -------------------------------------------------------------------
                bool borrowItem(int, Media*) {
                        // build Transaction object ('B', ptr);
                        // if ptr != NULL, get hash for id
                        // members[id].addTransaction(Transaction)
                        // ptr->decreaseCount()
                }


                // ------------------------------returnItem---------------------------
                // returnItem: applies a return transaction for a particular customer
                //                          + media item.
                // preconditions: customer exists, 0 < id < 10,000
                //                             media item exists.
                // postconditions: customer transaction history + item stock updated.
                // -------------------------------------------------------------------
                bool returnItem(int, Media*) {
                        // build Transaction object ('R', ptr);
                        // if ptr != NULL, get hash for id
                        // members[id].addTransaction(Transaction)
                        // ptr->increaseCount()
                }


                // --------------------------displayHistory-------------------------
                // displayHistory: outputs the transaction history for a particular
                //                                  customer.
                // preconditions: customer exists, 0 < id < 10,000
                // postconditions: customer transaction history, if any, is output.
                // -----------------------------------------------------------------
                bool displayHistory(int) const {
                        // get id hash
                        // members[hash].displayHistory();
                }


                // -------------------------------display-----------------------------
                // display: displays all customers and media items in the store,
                //                       + name if a named store.
                // preconditions: none.
                // postconditions: all Store info output to console.
                // -------------------------------------------------------------------
                void display() const;
```

```
        // --------------------------displayCustomers-----------------------
        // displayCustomers: outputs all customers.
        // preconditions: none.
        // postconditions: all customers, if any, are output as
        //                                #### fname lname
        // -------------------------------------------------------------------
        void displayCustomers() const;


        // --------------------------displayInventory-----------------------
        // displayInventory: outputs all media items.
        // preconditions: none.
        // postconditions: all media items, if any, are output as they
        //                                appeared in input.
        // -------------------------------------------------------------------
        void displayInventory() const;


        // -----------------------------getName-----------------------------
        // getName: obtains store name, if any.
        // preconditions: none.
        // postconditions: store name returned.
        // -------------------------------------------------------------------
        string getName() const;


        // --------------------------customerExists-------------------------
        // customerExists: determines whether a customer ID exists in the
        //                                Store.
        // preconditions: 0 < id < 10,000
        // postconditions: true if id found, otherwise false.
        // -------------------------------------------------------------------
        bool customerExists(int) const;


        // ----------------------------getCustomer--------------------------
        // getCustomer: obtains customer info matching id.
        // preconditions: 0 < id < 10,000
        // postconditions: customer returned, if found.
        // -------------------------------------------------------------------
        Customer getCustomer(int) const;

    private:
        string name; // title of business
        const static int MAX_CUSTOMERS = 10000;
        const static int MAX_GENRES = 53;              // 2x alphabet, closest prime

        Customer members[MAX_CUSTOMERS]; // customer hash table
        BinTree inventory[MAX_GENRES];   // media inventory

        // -----------------------------hash--------------------------------
        // hash: obtains a hash based on customer name, for the Customer hash
        //               table + genre char for the Media tree.
        // preconditions: string is nonempty.
        // postconditions: Customer hash returned.
        // -------------------------------------------------------------------
        int hash(string) const;
};
```

```
// --------------------------------customer.h--------------------------------
// Adam Ali / Hyosang Park CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// --------------------------------------------------------------------------
// Describes the ADT Customer such that a record of id #, first/last name, and
// transaction history regarding media borrows/returns and history output
// requests can be recorded.
//
// This customer class deals with every customer-related information.
// For example, id and name information of each customer are used to identify
// each customer and history vector is used to store each customer's
// transaction history. Therefore, we can know what items each customer
// borrowed or returned.

// --------------------------------------------------------------------------
// Functionality includes:
//              - create a Customer
//      - copy a Customer
//      - destruct a Customer
//      - retrieve id/first/last
//      - display Customer info/history
//      - add a new Transaction to the Customer transaction historu
//      - evaluat whether two Customers are equivalent

#pragma once

#include <string>
#include <vector>
#include "transaction.h"
using namespace std;

class Customer {
        public:
                // -------------------------------Customer-----------------------------
                // Customer: creates a unique Customer.
                // preconditions: 0 <= id < 10,000, strings non-empty
                // postconditions: specified Customer created.
                // --------------------------------------------------------------------
                Customer(int, string, string);


                // -------------------------------Customer-----------------------------
                // Customer: creates a copy of the Customer.
                // preconditions: none.
                // postconditions: a copy of the other Customer is created.
                // --------------------------------------------------------------------
                Customer(const Customer&);


                // -------------------------------Customer-----------------------------
                // ~Customer: destructs the Customer and frees any assoc. memory.
                // preconditions: none.
                // postconditions: any assoc. memory is freed, object inaccessible.
                // --------------------------------------------------------------------
                ~Customer();


                // --------------------------------getID-------------------------------
                // getID: obtains unique ID.
                // preconditions: none.
                // postconditions: unique ID returned.
                // --------------------------------------------------------------------
                int getID() const;
```

```
// ------------------------------getFirst-------------------------------
// getFirst: obtains the first name.
// preconditions: none.
// postconditions: first name returned.
// ---------------------------------------------------------------------
string getFirst() const;


// ------------------------------getLast--------------------------------
// getFirst: obtains the last name.
// preconditions: none.
// postconditions: last name returned.
// ---------------------------------------------------------------------
string getLast() const;


// ------------------------------display--------------------------------
// display: outputs unique ID + f/lname
// preconditions: none.
// postconditions: identity output to console.
// ---------------------------------------------------------------------
void display() const;


// -------------------------displayHistory---------------------------
// displayHistory: outputs all Transaction history.
// preconditions: none.
// postconditions: all available Transaction history is output.
// ---------------------------------------------------------------------
void displayHistory() const {
        // for all transactions in vector
                // transaction.display();
}


// -------------------------addTransaction---------------------------
// addTransaction: add a Transaction to history as most recent.
// preconditions: none.
// postconditions: history vector includes Transaction as latest.
// ---------------------------------------------------------------------
bool addTransaction(Transaction) {
        // history.push_back(Transaction);
}


// ----------------------------operator==-------------------------------
// operator==: checks whether two customers are exactly same by using
//             each customer's id number, because every customer gets
//             a unique id number.
// preconditions: none.
// postconditions: true if identical, false otherwise.
// ---------------------------------------------------------------------
bool operator==(const Customer&) const;


// ----------------------------operator!=-------------------------------
// operator!=: checks whether two customers are not the same by using
//             each customer's id number, because every customer gets
//             a unique id number.
// preconditions: none.
// postconditions: true if different, false otherwise.
// ---------------------------------------------------------------------
bool operator!=(const Customer&) const;
```

```
        private:
                int id;                                                 // customer unique id #
                string fname;                                   // first name
                string lname;                                   // last name
                vector<Transaction> history;    // transaction history
};
```

```
// -------------------------------transaction.h-----------------------------
// Adam Ali / Hyosang Park CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// -------------------------------------------------------------------------
// Describes the ADT Transaction such that any given transaction is processed
// for any given customer can be encapsulated to be stored in a collection for
// records (history). Only the transaction type and associated Media item are
// recorded, the customer is tracked by virtue of the Transaction being in
// that customer's history.
//
// Holds what type of transaction [i.e. Borrow or Return] each customer did
// and what kind of media item is related with that transaction.
// -------------------------------------------------------------------------
// Functionality includes:
//              - create a new Transaction
//      - copy an exist Transaction
//      - destruct a Transaction
//      - retrieve the transaction type
//      - retrieve a ptr to the assoc. Media item
//      - display Transaction details

#pragma once

#include "media.h"

class Transaction {
        public:
                // ---------------------------Transaction-----------------------------
                // Transaction: creates a Transaction record.
                // preconditions: string is non-empty.
                // postconditions: Transaction created with specified trans type and
                //                                 ptr to assoc. Media item.
                // -------------------------------------------------------------------
                Transaction(string, Media*);


                // ---------------------------Transaction-----------------------------
                // Transaction: creates a copy of the Transaction.
                // preconditions: none.
                // postconditions: a copy of the Transaction is created.
                // -------------------------------------------------------------------
                Transaction(const Transaction&);


                // ---------------------------Transaction-----------------------------
                // ~Transaction: destructs the Transaction and frees any assoc. memory.
                // preconditions: none.
                // postconditions: any assoc. memory is freed, object inaccessible.
                // -------------------------------------------------------------------
                ~Transaction() {
                        // not responsible for deleting Media link
                }


                // -----------------------------getType-------------------------------
                // getType: obtains the trans type.
                // preconditions: none.
                // postconditions: trans type returned.
                // -------------------------------------------------------------------
                char getType() const;


                // -----------------------------getItem-------------------------------
```

```
                // getItem: obtains ptr to linked Media item.
                // preconditions: none.
                // postconditions: ptr to Media item returned, NULL if N/A.
                // ------------------------------------------------------------------
                Media* getItem() const;


                // -----------------------------display----------------------------
                // display: output the Transaction details to console.
                // preconditions: none.
                // postconditions: Transaction details, including trans type and Media
                //                                  information are output to console.
                // ------------------------------------------------------------------
                void display() const {
                        // output trans type
                        // item->display();
                }

        private:
                char type;      // transaction type (borrow, return or history)
                Media* item;    // ptr to assoc. Media item
};
```

```
// ---------------------------------media.h--------------------------------
// Adam Ali / Adam Mirza CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// ------------------------------------------------------------------------
// Describes the abstract ADT Media such that any kind of item can be held
// in the store. All Media items have stock counts and formats (DVD, etc).
// Derived classes of Media simply add any additional attributes that pertain
// to them. They are all equipped with operations to maintain the stocks,
// display the details, and make comparisons between one another.
//
// This class is an interface for any type of Media in a store. It serves as
// a guide for being able to keep track of the inventory of a Media and what
// format that media is. There are methods to checkout, return, and compare
// Media.
// ------------------------------------------------------------------------
// Functionality includes:
//              - create a Media item
//              - copy an exist Media item
//              - destruct a Media item
//              - retrieve the format
//              - retrieve stock counts
//              - increment/decrement stock counts
//              - display details
//              - comparison operators (==, <)

#pragma once

#include <string>
using namespace std;

class Media {
        public:
                // --------------------------------Media-----------------------------
                // Media: creates a Media object with specified stock and format.
                // preconditions: format non-empty string, e.g. "DVD"
                //                                 stock >= 0
                // postconditions: specified Media item created.
                // ------------------------------------------------------------------
                Media(int, string);


                // --------------------------------Media-----------------------------
                // Media: creates a copy of the Media item.
                // preconditions: none.
                // postconditions: a copy of the Media is created.
                // ------------------------------------------------------------------
                Media(const Media&);


                // -------------------------------~Media-----------------------------
                // ~Media: destructs the Media and frees any assoc. memory.
                // preconditions: none.
                // postconditions: any assoc. memory is freed, object inaccessible.
                // ------------------------------------------------------------------
                virtual ~Media();


                // ------------------------------getFormat---------------------------
                // getFormat: obtains format, e.g. "DVD", "VHS", etc.
                // preconditions: none.
                // postconditions: format is returned.
                // ------------------------------------------------------------------
                string getFormat() const;
```

```
        // ------------------------------getCount-----------------------------
        // getCount: obtains stock count available for check out.
        // preconditions: none.
        // postconditions: stock is returned.
        // -------------------------------------------------------------------
        int getCount() const;


        // ----------------------------increaseCount--------------------------
        // increaseCount: increments stock count, typically after a return.
        // preconditions: none.
        // postconditions: stock = stock + 1.
        // -------------------------------------------------------------------
        void increaseCount();


        // ----------------------------decreaseCount--------------------------
        // decreaseCount: decrements stock count, typically after a borrow.
        // preconditions: none.
        // postconditions: stock = stock - 1.
        // -------------------------------------------------------------------
        void decreaseCount();


        // -------------------------------display-----------------------------
        // display: outputs Media details, pertaining to particular style.
        // preconditions: none.
        // postconditions: Media details output to console.
        // -------------------------------------------------------------------
        virtual void display() const = 0;


        // -------------------------------operator==--------------------------
        // operator==: determines if both Media items are identical.
        // preconditions: none.
        // postconditions: true if identical, otherwise false.
        // -------------------------------------------------------------------
        virtual bool operator==(const Media&) const = 0;


        // -------------------------------operator<---------------------------
        // operator<: determines if this Media precedes the other Media.
        // preconditions: none.
        // postconditions: true if preceding, otherwise false.
        // -------------------------------------------------------------------
        virtual bool operator<(const Media&) const = 0;

    private:
        int count;              // amount available for checkout
        string format;  // DVD, VHS, Book, Magazine, etc.
};
```

```
// ---------------------------------movie.h--------------------------------
// Adam Ali / Omar Aguirre CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// ------------------------------------------------------------------------
// Describes the abstract ADT Movie such that any genre/category of movie can
// be held in the store. All movies have a director, title, and release year.
// Derived classes of Movie add whatever attributes pertain to their genre,
// but all are equipped with operations to retrieve their attributes, display
// details, and make comparisons. All derivatives will have a const string
// to record their genre, which should be identical to the class name.
// ------------------------------------------------------------------------
// Functionality includes:
//              - create a Movie item
//      - copy an existing Movie item
//              - destruct a Movie item
//              - retrieve attributes
//              - display details
//              - comparison operators (==, <)

#pragma once

#include <string>
#include "media.h"
using namespace std;

class Movie : public Media {
        public:
                // --------------------------------Movie-----------------------------
                // Movie: creates a specified Movie.
                // preconditions: strings are nonempty, year >= 0.
                // postconditions: a Movie is created with specified fields.
                // ------------------------------------------------------------------
                Movie(string, string, string, int);


                // --------------------------------Movie-----------------------------
                // Movie: creates a copy of the Movie.
                // preconditions: none.
                // postconditions: a copy of the Movie is created.
                // ------------------------------------------------------------------
                Movie(const Movie&);


                // -------------------------------~Movie-----------------------------
                // ~Movie: deletes all assoc. memory. since variables for this class are
                //                 statically alloc'd, this destructor enables an appropriate
                //                 order for constructor calls for child classes.
                // preconditions: none.
                // postconditions: all assoc. memory de-alloc'd.
                // ------------------------------------------------------------------
                virtual ~Movie();


                // ----------------------------getDirector-------------------------
                // getDirector: obtains the director.
                // preconditions: none.
                // postconditions: director is returned.
                // ------------------------------------------------------------------
                string getDirector() const;


                // ----------------------------getTitle----------------------------
                // getTitle: obtains the title.
```

```
        // preconditions: none.
        // postconditions: title is returned.
        // ------------------------------------------------------------------
        string getTitle() const;


        // -----------------------------getYear-----------------------------
        // getYear: obtains the year.
        // preconditions: none.
        // postconditions: year is returned.
        // ------------------------------------------------------------------
        int getYear() const;


        // -----------------------------display-----------------------------
        // display: output Movie details to console. implementation left to
        //          children.
        // preconditions: none.
        // postconditions: Movie remains unchanged.
        // ------------------------------------------------------------------
        virtual void display() const = 0;


        // ----------------------------operator==----------------------------
        // operator==: compares both Media objects to check if they are equal.
        //                          implemented by child classes.
        // preconditions: none.
        // postconditions: true if identical, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator==(const Media&) const = 0;


        // ----------------------------operator<-----------------------------
        // operator<: compares this Media object to check if it precedes the
        //                         other. implemented by child classes.
        // preconditions: none.
        // postconditions: true if preceding, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator<(const Media&) const = 0;

    private:
        string director;
        string title;           // film title
        int year;                       // year of release

};
```

```
// ---------------------------------classic.h-------------------------------
// Adam Ali / Omar Aguirre CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// -------------------------------------------------------------------------
// Describes the ADT Classic such that a particular Movie genre can not only
// maintain a record of director, title and year (as all Movie genres do), but
// to also record the first + last name of the actor and the release month.
//
// Classic is a child class derived from Movie.
// -------------------------------------------------------------------------
// Functionality includes:
//              - create a Classic item
//      - copy an existing Classic item
//              - destruct a Classic item
//              - retrieve attributes
//              - display details
//              - comparison operators (==, <)

#pragma once

#include <string>
#include "movie.h"
using namespace std;

class Classic : public Movie {
        public:
                // -------------------------------Classic---------------------------
                // Classic: creates a specified Classic item.
                // preconditions: strings nonempty.
                //                                1 <= month <= 12
                // postconditions: specified Classic is created.
                // -------------------------------------------------------------------
                Classic(string, string, int);


                // -------------------------------Classic---------------------------
                // Classic: creates a copy of the Classic.
                // preconditions: none.
                // postconditions: a copy of the Classic is created.
                // -------------------------------------------------------------------
                Classic(const Classic&);


                // ------------------------------~Classic---------------------------
                // ~Classic: frees all (static) alloc'd memory by engaging the right
                //                   sequence of destructors -- from child to parent.
                // preconditions: none.
                // postconditions: all (static) memory de-alloc'd.
                // -------------------------------------------------------------------
                virtual ~Classic();


                // ---------------------------getActorFName-------------------------
                // getActorFName: obtains the lead actor's first name.
                // preconditions: none.
                // postconditions: lead actor's first name returned.
                // -------------------------------------------------------------------
                string getActorFName() const;


                // ---------------------------getActorLName-------------------------
                // getActorLName: obtains the lead actor's last name.
                // preconditions: none.
```

```
        // postconditions: lead actor's last name is returned.
        // ------------------------------------------------------------------
        string getActorLName() const;


        // -----------------------------getMonth-----------------------------
        // getMonth: obtains the release year.
        // preconditions: none.
        // postconditions: release year is returned.
        // ------------------------------------------------------------------
        int getMonth() const;


        // -----------------------------display-----------------------------
        // display: outputs Classic details.
        // preconditions: none.
        // postconditions: details output to console.
        //                 Classic remains unchanged.
        // ------------------------------------------------------------------
        virtual void display() const;


        // -----------------------------operator==-----------------------------
        // operator==: determines if both items are identical, based on
        //                          attributes common to all Movie genres (unless a
        //                          comparison is made with another Classic, which includes
        //                          the additional attribs).
        // preconditions: none.
        // postconditions: true if identical, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator==(const Media&) const;


        // -----------------------------operator<-----------------------------
        // operator<: compares this Classic object to check if it precedes
        //                         the other.
        // preconditions: none.
        // postconditions: true if preceding, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator<(const Media&) const {
                //if titles are equal then compare directors
                //if directors are equal then check for year published
                // if this.title < Media.title
                        // return true
                // else if this.director < Media.director
                        // return true
                // else if this.year < Media.year
                        // return true

                // return false //None of the previous conditions were met
        }

    private:
        const string CATEGORY = "CLASSIC";      // genre
        string actorFName;                                      // lead actor first name
        string actorLName;                                      // lead actor last name
        int month;                                                      // month of release
};
```

```
// ----------------------------------Comedy.h-------------------------------
// Adam Ali / Omar Aguirre CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// -------------------------------------------------------------------------
// Describes the ADT Comedy such that a particular Comedy genre can not only
// maintain a record of director, title and year (as all Movie genres do), but
// to also record any other details that may be included later (now, none).
// The const string is the main differentiator between the other derivatives
// of Movie.

// Comedy is a child class derived from Movie.
// -------------------------------------------------------------------------
// Functionality includes:
//              - create a Comedy item
//      - copy an existing Comedy item
//              - destruct a Comedy item
//              - retrieve attributes
//              - display details
//              - comparison operators (==, <)

#pragma once

#include <string>
#include "movie.h"

class Comedy : public Movie {
public:
        // --------------------------------Comedy-----------------------------
        // Comedy: creates a Comedy item. No additional attributes.
        // preconditions: none.
        // postconditions: a Comedy item is created.
        // -------------------------------------------------------------------
        Comedy();


        // --------------------------------Comedy-----------------------------
        // Comedy: copies the Comedy item.
        // preconditions: none.
        // postconditions: a copy of the Comedy item is created.
        // -------------------------------------------------------------------
        Comedy(const Comedy&);


        // -------------------------------~Comedy-----------------------------
        // Comedy: frees all (static) alloc'd memory by engaging the right
        //                 sequence of destructors -- from child to parent.
        // preconditions: none.
        // postconditions: all (static) memory de-alloc'd.
        // -------------------------------------------------------------------
        virtual ~Comedy();


        // -------------------------------display-----------------------------
        // display: outputs Comedy details.
        // preconditions: none.
        // postconditions: Comedy details output to console.
        //                                 Comedy remains unchanged.
        // -------------------------------------------------------------------
        virtual void display() const;


        // ----------------------------operator==-----------------------------
        // operator==: determines if both items are identical, based on
```

```
        //                 attributes common to all Movie genres.
        // preconditions: none.
        // postconditions: true if identical, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator==(const Media&) const;


        // ----------------------------operator<---------------------------
        // operator<: compares this Comedy object to check if it precedes the
        //                        other.
        // preconditions: none.
        // postconditions: true if preceding, otherwise false.
        // ------------------------------------------------------------------
        virtual bool operator<(const Media&) const {
                //if titles are equal then compare directors
                //if directors are equal then check for year published
                // if this.title < Media.title
                        // return true
                // else if this.director < Media.director
                        // return true
                // else if this.year < Media.year
                        // return true

                // return false //None of the previous conditions were met
        }

private:
        const string CATEGORY = "COMEDY";
};
```

```
// ----------------------------------drama.h-------------------------------
// Adam Ali CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// ------------------------------------------------------------------------
// Describes the ADT Drama such that a particular Drama genre can not only
// maintain a record of director, title and year (as all Movie genres do), but
// to also record any other details that may be included later (now, none).
// The const string is the main differentiator between the other derivatives
// of Movie.
// ------------------------------------------------------------------------
// Functionality includes:
//              - create a Drama item
//      - copy an existing Drama item
//              - destruct a Drama item
//              - retrieve attributes
//              - display details
//              - comparison operators (==, <)

#pragma once

#include <string>
#include "movie.h"

class Drama : public Movie {
        public:
                // --------------------------------Drama-----------------------------
                // Drama: creates a Drama item. no additional attribs.
                // preconditions: none.
                // postconditions: a Drama item is created.
                // ------------------------------------------------------------------
                Drama();


                // --------------------------------Drama-----------------------------
                // Drama: creates a copy of the Drama.
                // preconditions: none.
                // postconditions: a copy of the Drama is created.
                // ------------------------------------------------------------------
                Drama(const Drama&);


                // -------------------------------~Drama-----------------------------
                // Drama: frees all (static) alloc'd memory by engaging the right
                //               sequence of destructors -- from child to parent.
                // preconditions: none.
                // postconditions: all (static) memory de-alloc'd.
                // ------------------------------------------------------------------
                virtual ~Drama();


                // -------------------------------display----------------------------
                // display: outputs Drama details to console.
                // preconditions: none.
                // postconditions: Drama details output to console.
                // ------------------------------------------------------------------
                virtual void display() const;


                // ----------------------------operator==----------------------------
                // operator==: determines if both items are identical, based on
                //                attributes common to all Movie genres.
                // preconditions: none.
                // postconditions: true if identical, otherwise false.
```

```
		// ------------------------------------------------------------------
		virtual bool operator==(const Media&) const;


		// -----------------------------operator<----------------------------
		// operator<: compares this Comedy object to check if it precedes the
		//                        other.
		// preconditions: none.
		// postconditions: true if preceding, otherwise false.
		// ------------------------------------------------------------------
		virtual bool operator<(const Media&) const {
			//if titles are equal then compare directors
			//if directors are equal then check for year published
			// if this.title < Media.title
				// return true
			// else if this.director < Media.director
				// return true
			// else if this.year < Media.year
				// return true

			// return false //None of the previous conditions were met
		}

	private:
		const string CATEGORY = "DRAMA";
};
```

```
// ------------------------------bintree.h---------------------------------
// Adam Ali CSS 343 A
// Created: 05/21/17
// Modified: 05/23/17
// ------------------------------------------------------------------------
// Describes the ADT BinTree such that a binary search tree can maintain
// an inventory of Media items for efficient retrieval by a client. There are
// only single instances of each Media item, stock counts are recorded
// internally.
// ------------------------------------------------------------------------
// Functionality includes:
//              - create an empty BinTree
//      - copy a BinTree
//      - destruct a BinTree
//      - insert a new Media item + associated # copies
//      - retrieve a Media item
//      - display the contents of the BinTree (inorder traversal)

#pragma once

#include "media.h"

class BinTree {
        public:
                // ------------------------------BinTree-----------------------------
                // BinTree: creates an empty tree.
                // preconditions: none.
                // postconditions: an empty tree is created.
                // ------------------------------------------------------------------
                BinTree();


                // ------------------------------BinTree-----------------------------
                // BinTree: creates an identical deep copy of the other tree.
                // preconditions: none.
                // postconditions: a deep copy of the other tree is created.
                // ------------------------------------------------------------------
                BinTree(const BinTree&);


                // ------------------------------~BinTree----------------------------
                // ~BinTree: destructs the BinTree and frees any assoc. memory.
                // preconditions: none.
                // postconditions: any assoc. memory is freed, object inaccessible.
                // ------------------------------------------------------------------
                ~BinTree();


                /// ------------------------------insert----------------------------
                // insert: adds a new Media into the appropriate position within
                //         the tree.
                // preconditions: none.
                // postconditions: newly introduced Media inserted in appropriate
                //                 position.
                // ------------------------------------------------------------------
                bool insert(Media*, int);


                // ------------------------------retrieve----------------------------
                // retrieve: obtains the ptr that points to Media with matching
                //           Media.
                // preconditions: none.
                // postconditions: second Media points to located data, if match.
                //                 returns true if found, false otherwise.
```

```
                // ------------------------------------------------------------------
                bool retrieve(const Media& item, Media*& link);


                // -----------------------------display----------------------------
                // display: outputs tree contents using inorder traversal.
                // preconditions: tree is not empty.
                // postconditions: tree contents (all Media items) output to console.
                // ------------------------------------------------------------------
                void display() const;

        private:
                struct Node {
                        Media* item;
                        Node* left;
                        Node* right;
                };
                Node* root;

                // -----------------------------inorder-----------------------------
                // inorder: recursively performer inorder traversal, outputs
                //          console.
                // preconditions: Node != NULL.
                // postconditions: inorder traversal output to console, tree remains
                //                 unchanged.
                // ------------------------------------------------------------------
                void inorder(Node*) const;


                // --------------------------insertHelper---------------------------
                // insert: adds a new NodeData into the appropriate position within
                //         the tree using recursion.
                // preconditions: Node != NULL.
                // postconditions: newly introduced NodeData inserted in appropriate
                //                 position.
                // ------------------------------------------------------------------
                void BinTree::insertHelper(Media *&cur, Media *item, int copies);


                // -------------------------retrieveHelper--------------------------
                // retrieveHelper: recursively searches for Media and obtains a ptr
                //                 to encapsulating Node.
                // preconditions: Node != NULL.
                // postconditions: Node points to that which holds Media, if found.
                // ------------------------------------------------------------------
                void retrieveHelper(Node*, const Media&, Media*&);
};
```