Team ADS – Ahmad Yousif, Danny Banko, Aniley Sabi

Assignment 4 – Design

CSS 343

22 November 2020

Overview

This program creates a movie store which consists of an inventory of movies and a list of customers. Customers can borrow and return movies, as well as view what movies are in inventory and their history of transactions. With the movie inventory, clients can:

- Perform transactions with the movies (based on commands) in the inventory. The action items include borrowing (denoted as '**B**') and returning (denoted by '**R**').
- Output the current state of all the items in the inventory (denoted by '**I**')
- Output the history of transactions that the inventory has undergone since creation (denoted by '**H**')

The main program that will run and test this code will:

- Initialize a Store object *movieStore*
- Initialize a InventoryTree object to be added to *movieStore*
- Add the above movie objects to movieStore using addMovie() on each Movie object
- Add new Customer object to *movieStore's CustomerList,* then call the runBusiness() to perform different commands like borrow, and return.
- TubBusiness() will create Transaction objects for each call to the two methods, which will be stored in the respective Customer object's *transactionHistory()* field and the stores transactionHistory().

Our main will be very short, consisting of creating a Store object. The Store consists of a customer list, an inventory list, a factory to create inherited transactions and movies, and a general history of all transactions. Customers will keep track of their own transaction history. Our Transaction class has 3 child classes, Borrow, Return and History, Inventory is its own class, *not* a child class of Transaction. Our store will also have an InventoryTree which holds all the different movie objects. Movie is the parent class of 3 types of movies: Classic, Comedy, and Drama. Factory will be used to determine what type of

Movie or Transaction is created based off the data files “lab4movies.txt” and ”lab4commands.txt”.

The relationships and interactions between the classes in this program are as follows:

- **Aggregation (Aggregate --> Source)**
  - Transaction --> Customer

  Customer has a Stack of their transaction history

  - Transaction --> Factory

  Factory determines which type of Transaction to create

  - Customer --> CustomerHash

  CustomerHash holds all the Customers

  - CustomerHash --> Store

  Store holds the CustomerHash

  - Factory --> Store

  Store holds the Factory

  - Movie --> Factory

  Factory determines which Movie object to create

  - InventoryTree --> Store

  Store holds the InventoryTree

  - Movie --> InventoryTree

  InventoryTree holds all the movies

- **Inheritance (Inheritor --> Source)**
  - Classics ---> Movie

  Inherits all behavior and characteristics of the Movie object, but with a unique denotation (‘C’), as well as a unique constructor and destructor. This class also contains extra fields: majorActor and releaseDate

  - Comedy ---> Movie

  Inherits all behavior and characteristics of the Movie object, but with a unique denotation (‘F’), as well as a unique constructor and destructor.

- Drama ---> Movie

Inherits all behavior and characteristics of the Movie object, but with a unique denotation ('D'), as well as a unique constructor and destructor.

- Borrow ---> Transaction

Inherits all behavior and characteristics of the Transaction object, but with a unique denotation ('B'), as well as a unique constructor and destructor.

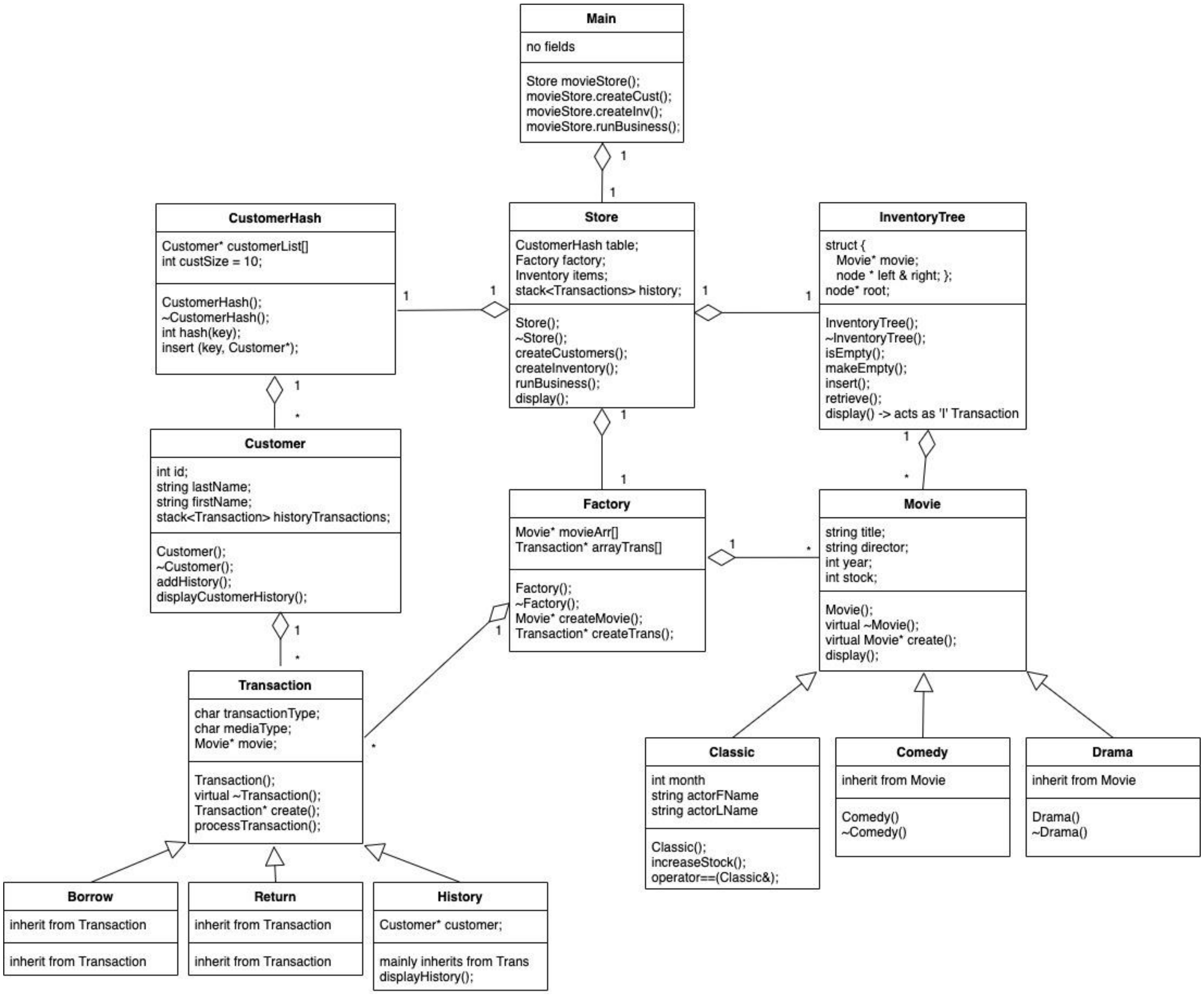
- Return ---> Transaction

Inherits all behavior and characteristics of the Transaction object, but with a unique denotation ('R'), as well as a unique constructor and destructor.

- History ---> Transaction

Inherits all behavior and characteristics of the Transaction object, but with a unique denotation ('H'), as well as a unique constructor and destructor. History also contains a pointer to the Customer class to access individual customer transaction history.

UML Class Diagram



Class Descriptions

Main:

```
//---------------------------------------------------------------------------
// Main
// Implementation:
// -- Responsible for building and running the movie business. Will create the
//    store and then perform all functions to run the business.
// -- Opens files and checks to make sure they can be read
// -- Calls all store methods to build and run the store.
//---------------------------------------------------------------------------
#include "store.hpp"
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, const char * argv[]) {
   ifstream infile1("data4customers.txt");  // customer file
   if (!infile1) {
      cout << "Error reading customers file)" << endl;
      return 1;
   }
   ifstream infile2("data4movies.txt");     // movie file
   if (!infile2) {
      cout << "Error reading movies file" << endl;
      return 1;
   }
   ifstream infile3("data4commands.txt");   // transaction file
   if (!infile3) {
      cout << "Error reading commands file" << endl;
      return 1;
   }
   // builds and runs business
   Store movieStore;
```

```
    movieStore.buildCustomerList(infile1);
    movieStore.buildInventory(infile2);
    movieStore.processTransactions(infile3);
    return 0;
}
```

Store:

```
//---------------------------------------------------------------------------
// Store
// Implementation:
// -- This class is the driver for all the other classes in this program.
//    Store is responsible for running the movie business and calling all
//    necessary functions to build and run the business from lower level
//    classes.
// Assumptions:
// -- Customer data files are formatted properly
// -- Transaction data files are formatted properly however, data can be
//    incorrect
// -- Transaction data files are formatted properly however, data can be
//    incorrect
//---------------------------------------------------------------------------
#ifndef store_hpp
#define store_hpp
#include <iostream>
#include <fstream>
#include <stack>
using namespace std;

class Store {
public:
    // constructors/destructors
    Store();  // constructor
```

```
  ~Store(); // destructor
  void buildCustomerList(ifstream&);   // populates the customers
  void buildInventory(ifstream&);      // populates inventory tree
  void processTransactions(ifstream&); // runs the transactions
  // testing
  void displayCustomers() const;  // used for testing, shows all customers
  void displayInventory() const;  // used for testing, shows all movies
  // retrieving customers
  bool findCustomer(int) const;    // tells us if the customer exists
  Customer& getCustomer(int) const; // returns the customer for transaction
private:
  CustomerHash table;   // hashTable for customers
  Factory factory;      // factory to choose which movie or
                        //   Transaction to build
  InventoryTree inventory;  // holds all movie items
  stack<Transaction> allTransactions;  // holds all store transactions
};
#endif
```

Factory:

```
//---------------------------------------------------------------------------
// Factory
// Implementation:
// -- Responsible for choosing which Movie class or Transaction class to
//    create. The factory is just used to create
// Assumptions:
// -- Data files are formatted properly, although data can be incorrect, the
//    lower level classes will check the validity of the Transaction methods
//    or the Movie type
//---------------------------------------------------------------------------
#ifndef factory_hpp
#define factory_hpp
```

```
#include <iostream>
#include <fstream>
using namespace std;
class Factory {
public:
  // constructors / destructors
  Factory();
  ~Factory();
  // creates the proper type of Transaction or Movie
  Movie* createMovie(char, ifstream&);
  Transaction* createTransaction(char, ifstream&);
private:
  Movie* storeMovies[26]; // assumes 26 possible movie types because 26
                          // letters in the alphabet
  Transaction* storeTransactions[26] // assumes 26 (same reason as above)
};
#endif
```

CustomerHash:

```
//---------------------------------------------------------------------------
// CustomerHash
// Implementation:
// -- This class is responsible for storing Customers for quick look up and
//    retrieval. This class will find a proper index to store a customer and
//    will insert Customers to the proper index based on their Customer ID
// Assumptions:
// -- Customers hashValues will be decided based on the customers 4-digit id.
// -- id's will be unique
// -- if there are multiple values at the same list index, they will be
//    connected by a linked list
//---------------------------------------------------------------------------
#ifndef customerHash_hpp
```

```
#define customerHash_hpp
#include <iostream>
using namespace std;
class CustomerHash {
public:
    CustomerHash();
    ~CustomerHash();
    int hash(const int) const;      // gives index value for lookup/insert
    bool insert(int key, Customer*); // will insert pointer Customer
    void display() const;          // displays customer
private:
    // used in the case multiple customers share same index (linked list)
    struct Node {
        Node* next;
        Customer* data;
    };
    Node* head;
    Customer* customerList[10000];   // hashTable for customers, 10,000
                                  // max # of customers id's (0001-9999)
};
#endif
```

Customer:

```
//---------------------------------------------------------------------------
// Customer
// Implementation:
// -- Customers hold a 4-digit unique id a last name and a first name.
//    Customers will be held in a hash table for quick look up and retrieval.
//    Customers will also be in charge of holding their own Transaction
//    history with the use of STL stack for LIFO.
// Assumptions:
// -- The data file will be formatted correctly, as well as id's will be a
//    unique 4-digit number.
// -- Customers keep track of thier own Transaction history
// -- STL stack is used to LIFO display Transaction history
//---------------------------------------------------------------------------------
#ifndef customer_hpp
#define customer_hpp
#include <iostream>
#include <fstream>
#include <stack>  // used to store Customer's Transaction history
using namespace std;
class Customer {
public:
    // constructors / destructors
    Customer();
    Customer(ifstream);
    ~Customer();
    // history functions
    void addHistory(Transaction);
    void displayHistory() const;
    // getters for Customer retrieval and comparison
    int getID() const;
    string getLastName() const;
    string getFirstName() const;
    // operator overloads
    bool operator==(const Customer&) const;
    bool operator!=(const Customer&) const;
private:
    int id;
    string lastName;
    string firstName;
    stack<Transaction> history;
};
#endif
```

Transaction:

```
//---------------------------------------------------------------------------
// Transaction
// Implementation:
// -- This class is responsible for handling the transactions performed by
//    customers. Reads from "data4commands.txt and perform the transaction
//    depending on ActionType(B, R, I, H),
// Assumptions:
// -- Data file will be formatted correclty, however data may be incorrect
// -- Factory determines which type of transaction object is created
//---------------------------------------------------------------------------
#ifndef TRANSACTION_H
#define TRANSACTION_H
#include <fstream>
#include <iostream>
#include "customer.hpp"
using namespace std;
class Transaction {
public:
   // constructors / destructors
   Transaction();
   Transaction(ifstream&);
   ~Transaction();
   void printTransaction(); // print the current transaction
   // getters
   char getMediaType() const;  // get the media type F, C, D
   char getTransactionType() const; // get transaction type, B, R, I, H
   Movie* getMovie(Movie* movie);  // get movie
   void processTransaction(istream&); // process the transaction
protected:
   char transactionType;  // B, R, I, H - Borrow, Return, Inventory, History
   char mediaType;       // F, C, D  Funny, Classic, or Drama
```

```
    Char item; // dvd, vcr, etc.
};
#endif
```

Borrow:

```
//---------------------------------------------------------------------------
// Borrow
// Implementation:
// -- This class is responsible for updating the stock: decrease the stock by
//    1. It is Subclass of the Transaction class, get actionType B and update
//    the stock
// Assumptions:
// -- inherits all behavior from Transaction parent class, although it has
//    a unique constructor and destructor
//---------------------------------------------------------------------------
#ifndef borrow_hpp
#define borrow_hpp
#include <fstream>
#include <iostream>
#include "transaction.hpp"
using namespace std;
class Borrow : public Transaction {
public:
    // constructors / destructors
    Borrow();
    ~Borrow();
    void display() const; // this is to print the borrow info
};
#endif
```

Return:

```
//---------------------------------------------------------------------------
// Return
// Implementation:
```

```
// -- This class is responsible for updating the stock: increase the stock by
//    1. It is the Subclass f the Transaction class, get actionType R and
//    update the stock
// Assumptions:
// -- Inherits all behavior from Transaction parent class, although it has
//    a unique constructor and destructor
//---------------------------------------------------------------------------
#ifndef returnItem_hpp
#define returnItem_hpp
#include <fstream>
#include <iostream>
#include "transaction.hpp"
using namespace std;
class Return : public Transaction {
public:
   Return();
   ~Return();
   void display() const;      // to print the return info
};
#endif
```

History:

```
//---------------------------------------------------------------------------
// History
// Implementation:
// -- This class is responsible for storing the history of customers
//    transactions with movie type
// Assumptions:
// -- Inherits all behavior from Transaction parent class, although it has
//    a unique constructor and destructor
// -- display() makes a call to Customer displayHistory() method
//---------------------------------------------------------------------------
#ifndef history_hpp
```

```
#define history_hpp
#include <fstream>
#include <iostream>
#include "transaction.hp"
using namespace std;
class History : public Transaction {
public:
    History();
    ~History();
    void display() const;
};
#endif
```

InventoryTree:

```
//---------------------------------------------------------------------------
// InventoryTree - Header File
// Implementation:
// -- the internal representation of a binary search tree. Constructors, accessors,
//    and other operators and functions have been defined and overloaded for BST
//    manipulation in this program.
// Assumptions:
// -- Data file will be formatted correctly, although data may be incorrect
// -- Factory determines which type of movie object is created
//---------------------------------------------------------------------------
#ifndef inventoryTree_hpp
#define inventoryTree_hpp
#include <iostream>
#include "movie.hpp"
using namespace std;
class InventoryTree {
public:
    // constructors / destructors
    InventoryTree();
    InventoryTree(const InventoryTree &obj);
    ~InventoryTree();
    bool isEmpty() const;  // checks if its empty
    void makeEmpty();      // will empty tree in destructor
    bool insert(Movie* obj);  // inserts movie object
    // retrieves a Movie from the tree
    bool retrieve(const Movie &data, Movie* &retrieveData) const;
    // will display the BST of movies
    void display() const;
private:
    // ALL HELPER FUNCTIONS HERE
    // used as nodes for the tree
    struct Node {
        Movie* data;  // pointer to data object
        Node* left;   // left subtree pointer
        Node* right;  // right subtree pointer
    };
    Node* root;   // root of the tree
};
#endif
```

Movie:

```
//---------------------------------------------------------------------------
// Movie - Header File
// Implementation:
// -- This class is responsible for handling the movies available in the store.
//    It reads from "data4movies.txt" and creates a movie object depending on
//    depending on movie genre (F, D, or C) and other inputted movie information.
// Assumptions:
// -- Data file will be formatted correctly, although data may be incorrect
// -- Factory determines which type of movie object is created
//---------------------------------------------------------------------------
#ifndef movie_hpp
```

```
#define movie_hpp
#include <fstream>
#include <iostream>
using namespace std;
class Movie {
public:
   // constructor / destructor
   Movie();
   ~Movie();
   // Creates a movie object of the appropriate genre based on input
   Movie* create(istream& genre);
   void display();  // prints Movie
private:
   string title;    // movie title
   string director; // director name
   int year;        // release year
   int stock;       // amount in stick
};
#endif
```

Classic:

```
//---------------------------------------------------------------------------
// Classic
// Implementation:
// -- This class is responsible for handling the classic movies available in
//    the store. Classic inherits the public variables and methods from Movie.
// Assumptions:
// -- Data file will be formatted correctly, although data may be incorrect
// -- Factory determines which type of movie object is created
//---------------------------------------------------------------------------
#ifndef classic_hpp
#define classic_hpp
#include <fstream>
```

```
#include <iostream>
#include "movie.hpp"
using namespace std;
class Classic : public Movie {
public:
   // constructor / destructor
   Classic();
   ~Classic();
private:
   // Constant character indicating the movie's genre is classic
   const char GENRE_CLASSIC = 'C';
   string actorFName;   // actor's first name
   string actorLName;   // actor's last name
   int month;           // release month
};
#endif
```

Comedy:

```
//---------------------------------------------------------------------------
// Comedy - Header File
// Implementation:
// -- This class is responsible for handling the comedy movies available in
//    the store. Comedy inherits the public variables and methods from Movie.
// Assumptions:
// -- Data file will be formatted correctly, although data may be incorrect
// -- Factory determines which type of movie object is created
//---------------------------------------------------------------------------
#ifndef comedy_hpp
#define comedy_hpp
#include <fstream>
#include <iostream>
#include "movie.hpp"
using namespace std;
```

```
class Comedy: public Movie {
public:
   // constructor / destructor
   Comedy();
   ~Comedy();
private:
   // Constant character indicating the movie's genre is comedy
   const char GENRE_COMEDY = 'F';
};
#endif
```

Drama:

```
//---------------------------------------------------------------------------
// Drama - Header File
// Implementation:
// -- This class is responsible for handling the drama movies available in
//    the store. Drama inherits the public variables and methods from Movie.
// Assumptions:
// -- Data file will be formatted correctly, although data may be incorrect
// -- Factory determines which type of movie object is created
//---------------------------------------------------------------------------
#ifndef drama_hpp
#define drama_hpp
#include <fstream>
#include <iostream>
#include "movie.hpp"
using namespace std;
class Drama : public Movie {
public:
   // constructor / destructor
   Drama();
   ~Drama();
private:
  // Constant character indicating the movie's genre is drama
  const char GENRE_DRAMA = 'D';
};
#endif
```

Classes we will not implement but we are using:

STL Stack:

We are using a stack to keep track of the history of transactions for each customer and the entire store. Stack will allow us to display them in LIFO like the assignment specifies.

Pseudo Code:

```
Void buildCustomerList(ifstream& customers) {
        If (customers) {
                While(!customers.eof()) {
                Read customers into Customer(ifstream&);
                Get customers hash value;
                Insert into hashtable;
                }
        } else  {
                Error message
        }
}


Void buildInventory(ifstream& movies) {
        If (movies) {
                        Char type;
                While (!movies.eof) {
                        Char type;
```

```
                    Movies >> type;
                    Use factory to create proper movie object;
                    Insert into Inventory BST
                }
            }
        }
Void processTransactions(ifstream& commands) {
    If (commands) {
        While (!commands.eof) {
            Char commandType;
            Int id;
            Commands >> commandType;
            Commands >> id;
            Use factory to create proper transaction type;
            Search customer hash table to find customer;
            If customer exists {
                Store a pointer to the customer;
                Call executeTransaction() to perform the transaction
                // updates inventory and customer stack
            } else {
                Cout << Error message;
            }
        }
    }
}
```

Additional Notes [to extend specifications]:

- New genres could easily be added by creating another child class to the parent Movie and then using factory to create that object
- Items could be added by adding a parent class to Movie called Item, which could have other children classes such as Music.
- Other Transactions could be added by creating and additional child Class of Transaction and adding that class to the Factory transaction list
- Time/Due date could be an additional private data variable to the borrow items class.
- Store is its own class, so multiple store objects could be created. An additional class could also be added above store to hold different store objects.