**Class: CSS343**

**Assignment: Movie Rental Store Inventory Tracking System Design**

**Design Group 4: Aashima Mehta, Nick Shokri, Zachary Joseph Morrison, Zach Shim, Xue Guan/Cheryl**

**Date: 08/07/20**

**Overview:**

Program 4 is a movie inventory tracking system for movie rental stores such as Blockbusters. The program keeps track of customer information and the stock number of each movie in the store. The program takes movie data, customer data, and transaction data from text files and processes them. Customer information and movie information will be stored using a hash table. If any invalid commands are read from the text files, an error message will be displayed.

**Description of classes:**

* Movie: A general type of movie that specific genres of movies (Comedy, Drama, Classics) inherit from.
  + Abstract class
* Comedy: A specific genre of Movie.
  + Derived class from Movie
* Drama: A specific genre of Movie.
  + Derived class from Movie
* Classics: A specific genre of Movie.
  + Derived class from Movie
  + Read  majorActor releaseMonth insteading of release year
* movieFactory: creates a movie object based on type
* Store: A management class in charge of building movies, customers, transactions and get inventory and customer information
* Transactions:A general type of transactions that specific genres of transactions(Borrow, Return, Inventory and History) inherit from.
  + Abstract class
* Borrow: A class that represents an operation to remove the copies of Movies from the Store.
  + Derived class from transaction and stock need to be decreased
* Return: A class that represents an operation to add the copies of movies to the Store.
  + Derived class from transaction and stock need to be increased
* Inventory: A class that represents an operation to display all items currently in the Store.
  + Derived class from transaction
* History: A class that represents an operation to display all transaction history for each customer
  + Derived class from transaction
* transactionFactory: Factory class for creating a  transaction subject.
* HashTable: Stores customer information and movie information in a hash table.
* Customer: Stores information for an individual customer such as first name, last name and id.

**Description of main:**

int main() {

// open streams

ifstream infileM("data4movies.txt");

if (!infileM)  return 1;

ifstream infileC("data4customers.txt");

    if (!infileC) return 1;

ifstream infileT("data4commands.txt");

if (!infileT) return 1;

// create store object and execute methods  
Store s;

s.buildMovies(infileM);

s.buildCustomer(infileC);

s.processTransactions(infileT);

return 0;

}

**Pseudo Code Flow-Chart**

-------------------------------------------------------------------------------------------------------------------------------

void **Store::buildMovie**(ifstream& infile) {

           while(getline(infile, data)) {

                     Create new movie pointer; // Example Movie\* m;

                     Bool success = MovieFactory::createMovies(newMovie, data);

                     If(not successful) continue;

                     Add movie to hash table       // Example m->add();

           }

}

-------------------------------------------------------------------------------------------------------------------------------

// when initializing a new object, the Movie subclass’s string constructor takes in the data param // to set it’s data

Bool **MovieFactory::createMovies**(Movie\*& newMovie, string data) {

Switch(first letter of data) {

Case F:

Assign Movie\* movie to new Comedy object;

Break;

Case C:

Assign Movie\* movie to new Classics object;

Break;

Case D:

Assign Movie\* movie to new Drama object;

Break;

                    default:

             print error

newMovie = null;

                      return false;

}

// returns false if there is invalid data

// class Classic has different data members, but due to the polymorphism the

// correct setData will be called for whatever movie subclass object is initialized

return newMovie->setData(data);

}

-------------------------------------------------------------------------------------------------------------------------------

void **Store::buildCustomers**(Ifstream&) {

           while(not end of file) {

Customer\* c = new C;

                     c->setData(ifstream&);

                     If (setData was not successful) continue;

                     Add customer to hashtable;

           }

}

-------------------------------------------------------------------------------------------------------------------------------

void **Store::processTransactions**(ifstream&) {

String line;

while(getline(ifstream, line)) {

Transaction\* newTrans;

Bool success = TransactionFactory::createTransactions(newTrans, data);

If(not successful) continue;

newTrans->executeTransaction(this);

                        // Example Data: B 1234 D C 9 1938 Katherine Hepburn

}

}

-------------------------------------------------------------------------------------------------------------------------------

// creates a transaction depending on

Bool **TransactionFactory::createTransactions**(Transaction\*& newTrans, string data) {

           Switch(first char of data) {

                     Case ‘I’:

                               newTrans = Create new inventory object

                               break;

                     Case ‘H’:

      // uses History constructor

Discard first letter from string data param;

    newTrans = Create new history object(customerID from data parameter)

    break;

                     Case ‘B’:

                                newTrans = Create new borrow object;

                                break;

                     Case ‘R’:

                                newTrans = Create new return object;

                                break;

                     Default:

                                Print error message

                                newTrans = null

                                return false;

           }

           return true;

}

-------------------------------------------------------------------------------------------------------------------------------

// EXAMPLE FOR BORROW SUBCLASS

// Return::executeTransaction(Store\* s) would be identical, except in the last line of the method,

// you would perform m->addStock() instead of m->subtractStock();

void **Borrow::executeTransaction**(Store\* s) {

Customer\* c = Retrieve customer from hashtable(clientID);

retrieve movie from hashtable(m, s->getMovies);     // m is a data member of borrow

If(null) return;

m->subtractStock();

c.push\_back((this));

}

-------------------------------------------------------------------------------------------------------------------------------

// finds the movie in the movie hashtable depending on the key

void **Borrow::findMovie**(HashTable& movies) {

           m = Movies.retrieveMovie(movie key); // m is an internal data member for Borrow

}

-------------------------------------------------------------------------------------------------------------------------------

// HashTable is a database for customers and movies

<Type\*> **HashTable::retrieveItem**(int id) {

           Int hash = getHash(id);

           For each item in the linked list at array[hash] {

                     If (item->getId() == customerId) {

                                Return item;

                     }

           }

}

-------------------------------------------------------------------------------------------------------------------------------

Int **HashTable::getHash**(int id) {

return  id % size of array;

}