Assignment4 19

Tyler Quayle | Gytautas Jankauskas | Justin Rauvola

Overview

We have 4 main classes, RentalStore being the main class which contains a Vector of class customers. Customers contains variables for id and name. RentalStore also contains a list of Movies, movies is a superclass has 3 classes that inherit from it; Comedy, Classical, and Drama. When we read in our command file we will execute the commands as they are being in read in. Command has 4 derived classes; inventory, history borrow and return. Each of Commands derived classes handle their specific executions of the commands, for instance borrow will take 1 away from stock of the movie and put the history into the correct customer's history vector.

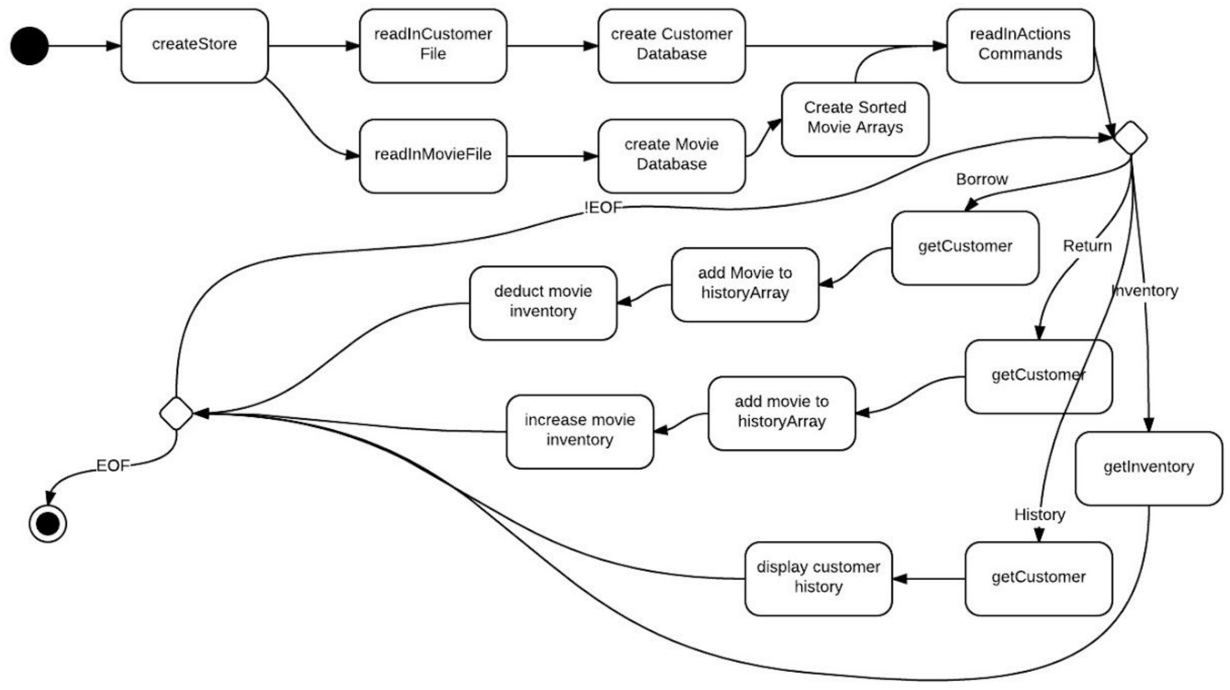
Problem Description

Creating a record system for a movie rental store, including a database of customers, each with their own history. And a database of movies, divided by movie type; classic, comedy, and drama being the current selection. The system must be designed in a way to be extensible for any future types of movie genres that the business may use may be easily created. The system must also be able to handle a sorting system that is unique to each type of movie genre the system handles.

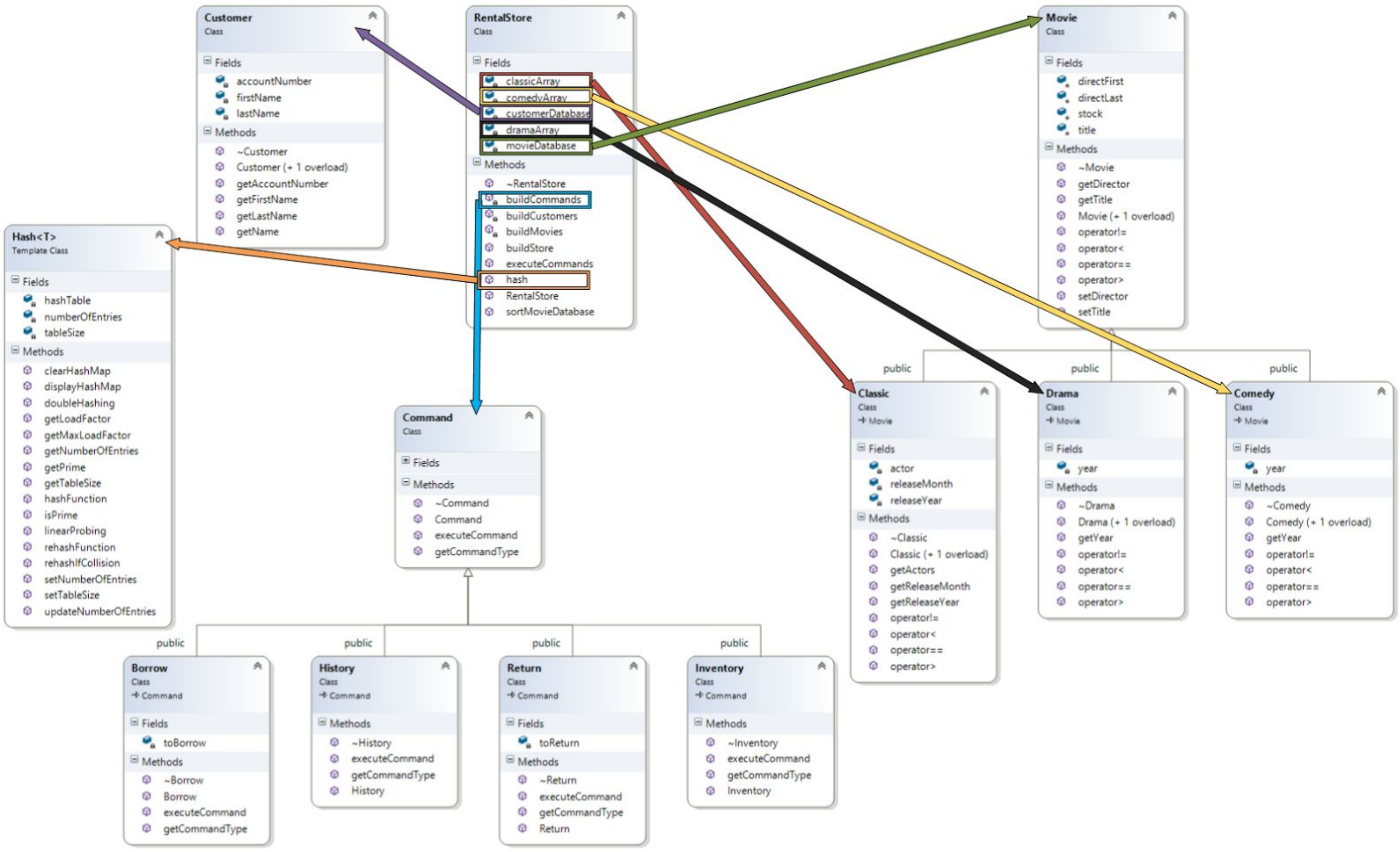
Assumptions and Design Considerations

We may assume that all data will be formatted in the text files, so reading in files will not have to cover special cases of missing commas or spaces. We have also designed the system to be extensible when it comes to types of movies, each genre of movie being a derived class of movies. We will use a hashtable to store the movie object. We will use quadratic probing for collision resolution of our hash table. We will start the hash table at a size of 7 and rehash as needed. We will also use a hashtable to store the customers. For customers, we will use linear probing for collision resolution of our hash table to show multiple types of collision resolution.

High Level Algorithms



Class Diagram



Class descriptions

RentalStore

This class represents a movie rental store. This class will hold the catalog of movie objects, and customer objects. Along with the duties of holding data, it is responsible for reading the file inputs. This class will accept file inputs needed to generate a movie object, a customer object, and will also process the execution of commands. With the duty of building a movie object, it will also be responsible for calling the hash object in order to store the movies in its movieDatabase(hash table). The customers will also be stored in a hash table. When movies need to be sorted, this class will also handle that responsibility by using the genre comparable operators and then outputting the information. (see pseudocode section)

Customer

This class represents a customer of a movie store. It is responsible for holding the customer's information: firstName, lastName, and accountNumber, and the customers history object. The duties of this class is to provide getter methods for the customers information.

Command

The command class mainly holds the account numbers and whether the command that is trying to execute is valid. It has a virtual executeCommand function that if the command is valid, goes down to the classes that inherit from it.

Borrow

If the command that the rental store passes is the command to borrow a movie, you go down to this class. This class executes the command using the account number inherited from the command class. The user borrows a movie, this adds this transaction to the customer's history and decrements the inventory of the movie borrowed.

History

The history, is the history of what movies a customer has borrowed and returned. When this class is entered, it is because a command is being executed. When this command is executed then a full list of the movies that the user borrowed and returned.

Return

When a user wants to return a movie then the return class is called. When it executes, it removes the movie that the user is returning from the inventory that they have and adds it back to the inventory of the items. It also adds to the history of the user that they returned the movie.

Inventory

The inventory will consist of the movies that the user holds onto at the moment. When they withdraw it shows that they have the movie and when they return it gets rid of that movie. The inventory helps the store see what user has what movies checked out. This also allows them to see if the customer is eligible to borrow or if they have outstanding fines due to not returning movies.

Movies

Movies is the superclass for the genres represented by the classes: Drama, Comedy, Classic. Since all movies genres contain the information for; *stock*, *director first name*, *director last name*, and *movie title* this information is stored in the Movie Class. Movie class has the basic get/set methods and overloaded constructors to deal with input from the file.

Drama

Drama is a subclass of Movies, the only piece of information this class handles is the year. Drama overloads operators >,<,==, != for use when sorting since all genres require different forms of sorting, drama needs to be sorted first by Director, then by title in cases of same name.

Comedy

Comedy is a subclass of Movies, the only variable it controls is *year*, comedy has overloads for <, >, ==, != to be used with sorting, since comedies need to be sorted by Title, and then by year, the compiler default operators wouldn't be able to handle these functions.

Classic

Classic is a subclass of Movies which contains both the *releaseMonth* and *releaseYear* that is required to be known by the classic movies. Classic also contains a pointer to an Array for actors, since multiple classics can appear in the file, there needs to be a way to contain multiple actors in the same instance. Classic overloads operators: <, >, !=, == to be able to handle sorting, classics must be sorted first by Major actor then by release date.

Pseudocode

```
// -------------------------------- RentalStore.h ----------------------------
        //**buildStore**This method will be responsible for accepting the movieData
        //                file and customer file. It will call buildMovies and buildCustomers
        //                in order to fill the databases.
        //**Pseudocode**
        //                read movieData file
        //                call buildMovies
        //                return boolean
        //                read customerData file
        //                call buildCustomers
        //                return boolean
        bool buildStore(std::string movieData, std::string customerData);

        //**executeCommands** This method will accept the file of commands
        //                it will call buildComands in order to process file.
        bool executeCommands(std::string commandData);

        //**sortMovieDatabase** This method will create a vector of the movie
        //                objects one for each genre. Then it will sort based off
        //                director, then title. For comedy the tie breaker wiil be
        //                the actor.
        //**pseudocode*
        //                use quicksort to sort based on specific class overloads
        //                output vector
        bool sortMovieDatabase();
```

```
	//**hashMovies** This method will use the hashClass's hashfunction
	//			to insert movies into the hashTable object.
	void hashMovies(std::string toHash);

private:
	//data member for holding the customers
	HashTable<Customer> customerDatabase;
	//pointer for accessing movies database
	Movie *movieDatabase;
	//HashTable for storing pointers to movies
	HashTable<Movie> moviesDataBase;
	//Arrays used for the sorting function
	vector<Comedy> comedyArray;
	vector<Drama> dramaArray;
	vector<Classic> classicArray;

	//**buildMovies** This method will process the moviesData file and return
	//				a boolean if successfully reads line of data.
	//**pseudocode**
	//	while contents in file
	//			get line up to comma, set movieType
	//			get line up to comma, set movieStock
	//			get line up to comma, set director
	//			start stringstream and set first name and last name
	//				while still text in ss, last name has a space
	//				temp last name, get rest of name, append space, append rest of name
to temp, lastname = temp
	//			get line up to comma, set movie title
	//			if movieType is comedy or drama, set yearRelease
	//			else if classic, set actor name, release month, release year.
	//			else not valid movieType output error
	//			create movie object
	//			call hash and store object
	//			store object in appropriate vector
	bool buildMovies(std::ifstream &inFile);

	//**buildCustomer** This method will process the customersData file and
	//			return a boolean depending on success
	//**pseudocode**
	//	for loop reading file
	//			read file, set id
	//			read file, set lastname
	//			read file, set firstname
	//			create customer object
	//			call hash and store
	bool buildCustomers(std::ifstream &inFile);
```

```
//**buildCommands** This method will process the commandsFile and return
//                  a boolean depending on success
//**pseudocode**
//          while contents in file
//                  read in commandtype
//                          if "i"
//                                  traverse hashtable output inventory
//                          else if "h"
//                                  get customer
//                                  output history
//                          else if "b"
//                                  if error on input
//                                          output error message
//                                  get customer
//                                  get movie from hash
//                                  if inventory <1
//                                          output message
//                                  else
//                                          update transaction history
//                                          reduce count of inventory by 1
//                          else if "r"
//                                  if error on input
//                                          output error message
//                                  get customer
//                                  get movie from hash
//                                  update transaction history
//                                  increase count of inventory
//                          else
//                                  output error message
bool buildCommands(std::ifstream &inFile);
```