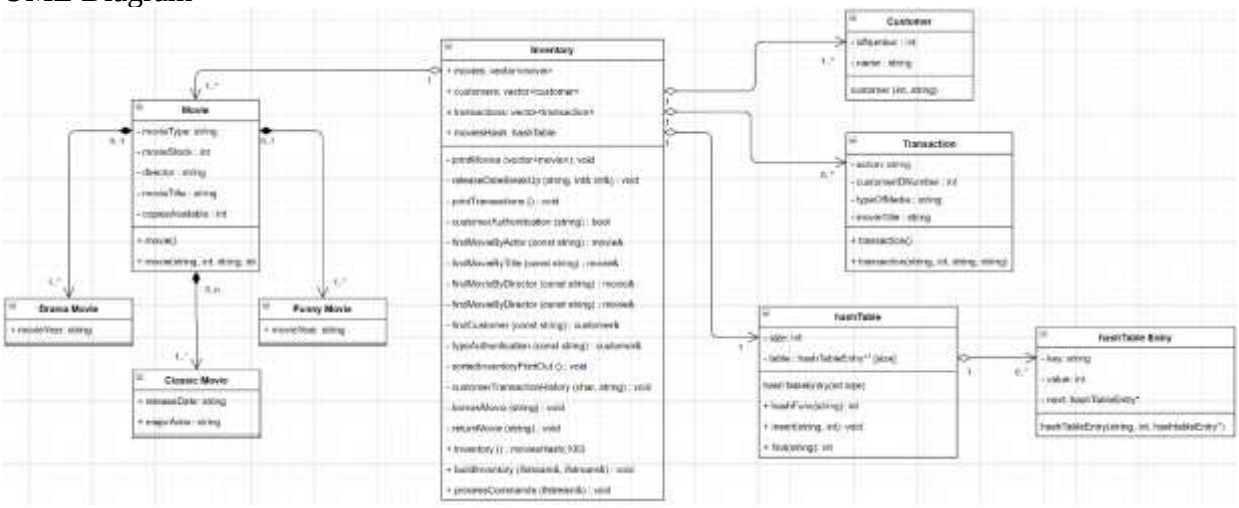
Overview:
The movie rental system will be made up of an inventory class that is responsible for maintaining and managing the movie inventory, customer list, and adding and removing transactions. The inventory system will also maintain a hash Table that stores movies titles and is used to find the index location in the movies inventory. This way, as the inventory grows, we maintain a near constant time complexity in our search functions of the movie inventory and can provide information needed in renting the movie, like is it available for borrowing. There will also be a transaction class this is responsible for recording all the details of a transaction. The main program will be responsible for opening three text files, one containing all movie inventory information, one with a list of customer information, and one that is a list of rentals and returns and the customers information associated with the transaction. Using the ifstream object, we create two objects that represent the movie file, customer file, and the commands. We then create an instance of a inventory database object and use its build inventory method and pass both ifstream objects in as parameters to build the inventory. After both lists have been created in the inventory, the main calls the inventory's method to process the commands file and update the database. There is a movie super class with three variations of genre subclasses, dramaMovie, comedyMovie, classicMovie. This inheritance structure is going to be in hierarchical format where the three subclasses individually inherit from the movie class. The movie class is maintained by the inventory class.

UML Diagram



Class Descriptions

Inventory Class

Private Data Members:

- vector<movie> movies'
- vector<customer> customers;
- vector<transaction> transactions;
- hashTable moviesHash;

Private Functions:

- vector<movie> movies
  /*
    A vector of movies objects that represents and holds the entire movie inventory
  */
- vector<customer> customers;
  /*
    A vector of customer objects that represents and holds the entire
      customer list
  */

- vector<transaction> transactions;
  /*
    A vector of transaction objects that represents and holds the entire customer transaction list
  */
- hashTable movieHash;
  /*
    A hash table of all current movie title names in the movie stores inventory. This is an indexing tool to help find the location of the movie object in the inventory so we can find information about its current state.
  */
- void printMovies(vector<movie> list)
  /*
  Helper function that outputs all movie information for all movies in the current inventory. It iterates through the inventory's movie vector list, printing out all information using the movies overloaded output operator. Used for testing the buildinventory(ifstream& movieFile, ifstream& customerFile) function.
  */
- void releaseDateBreakUp (string releaseDate, int& month, int& year)
  /*
    Helper function that helps deal with the classic movie titles that have a release date (month and year) by cutting the string in an appropriate way and converting both to integers, then storing that information in its own movie object
  */
- void printTransactions()
  /*
     Helper function that prints out all customer transactions. This Used for testing the transactions implementation.
  */
- bool customerAuthentication(string custID)
  /*
     Helper function that authenticates the customer IDs used when conducting a transaction with a customer. When a customer attempts to borrow a movie, the ID that is used and then compared against the vector of customer objects returning true if found and if not, output that the user ID used does not exist and return false.
  */
- movie& findMovieByTitle(const string movieTitle)
  /*
     The method will first use the hash classes' find() method to pass the string movie title. That function will return the index where the movie in located in

the movie vector to a local variable which is then used to return address location in memory that the movie exists.
*/

- movie& findMovieByDirector(const string director)
  /*
  Takes a directors string name and iterates over the movie vector, comparing the string to the movies director name. If it finds a match, return the current movies reference. If not, output that a movie with that directors' name is not available.
  */
- customer& findCustomer(const string custID)
  /*
  Converts the string custID to an integer then iterators through the customer list looking for the customer that matches the ID. If it matches the ID, it returns the address in memory that the customer object lives. If not, outputs that the customer doesn't exist.
  */
- bool mediaTypeAuthentication (const string mediaT)
  /*
  Takes the media type and makes sure it is a valid media type and returns true if so, and false if not.
  */
- bool movieTypeAuthentication (const string movieT)
  /*
  Takes the movie type and makes sure it is a valid media type and returns true if so, and false if not.
  */
- void sortedInvectoryPrintOut()
  /*
  Create three local vectors to store the ordered lists of classic, comedy, and drama movies. Create copy_if iterators that look to match the movie types and copy those movies to their respective classic, comedy, and drama vectors. Then call the objects sort function and pass it the beginning and end of the vector. Then print out the list of movies.
  */
- void customerTransactionHistory(char actionType, string ID)
  /*
  Takes a char actionType and a string ID to store the type of action being performed and the customer ID. Creates a vector of customer transactions that is the size of the size of the inventory classes' transactions vector. If the vector isn't empty, iterate through it and copy off the transactions that match the customers ID to the customer transactions vector. If the customers transactions vector isn't empty, print out the vector list of customer transactions. If not, print out that there are no transactions from this customer.
  */
- void borrowMovie(string information)
  /*
  Takes a string information that contains the borrow command, customerID, media type, movieType, release date, major actor.
  If the movie type is classic, the leftover string is parsed out into the released date month and year and the major actor. Then use the findMovieByActor function to return the location of the movie by passing the actors name. Then update the copies available. Then create a transaction documenting the details and add it to the inventory's transaction vector.

```
   If the movie type is a drama, parse out the movie title and director. And find
the movie by director to update the stock
   If the movie is a comedy, then parse out the movie title and title year and
find the movie with the mov name and return a pointer address to the location of
that movie. Use the pointer to update the copies available
*/
```

- `void returnMovie(string information)`

```
/*
   Takes in a string and to parse into customerID, movie information to find the
location of the movie in the inventory and return it so that number of available
copies can be updated. Using a vector iterator, search the movies inventory and
if found, create a transaction with the customer ID, and movieTitle and add the
transaction to the transaction vector.
*/
```

Public Functions:

- `void buildInventor(ifstream& movieFile, ifstream& customerFile)`

```
/*
   Class responsible for taking two text files and parsing them out into the
system. First, the movie file will be parsed so that the movie inventory will be
generated. Depending on the movieType, each parsing will be different because of
the formatting of the text file. Once all information is extracted from the
string and a new Movie object is created and pushed into movies vector. Then the
update hashTable by inserting the movies title as the key. Once the movies text
is finished, then parse the customer text file to build the customer database.
Parse out customer name and customer ID and pass into the customers constructor
to create a new customer. Then pass the customer to the customers vector
*/
```

- `void processCommands(ifstream& command)`

```
/*
   Responsible for parse a list of commands and executing them. Authenticate the
customerID and the action types before processing command.
*/
```

Customer Class

Private Data Members

- `int idNumber;`
- `string name;`

Public Functions:

- `customer(int idNumber, string name)`

```
/*
    Standard constructor passing the customers ID number and name
*/
```

Movie Class

Private Data Members:

- `string movieType       // classic, drama, comedy`
- `int movieStock  // tracks how many movies the company owns of that specific movie`
- `string director        // the directors name`
- `string movieTitle      // the movies title`
- `int copiesAvailable // Tracks the number available of any specific type of movie`

Public Functions:

- `friend class inventory`
  ```
  /*
     Grants the inventory access to the protected data members
  */
  ```
- `friend ostream& operator<< (ostream& os, const movie& m)`
  ```
  /*
     Allows movies to be output to the console and control the print formatting
  */
  ```
- `movie(string movieType, int movieStock, string director, string movieTitle)`
  ```
  /*
     Constructor
  */
  ```

Transaction Class

Private Data Members:

- `string action   //maintains whether the transaction was a barrow or a return`
- `int customerIDnumber;        //maintains customerID`
- `string typeOfMedia;          //maintains type of media`
- `string movieTitle;           //maintains movie title`

Public Functions

- `friend ostream& operator<< (ostream& os, const transaction& t)`
  ```
  /*
     Allows transactions to be output in a controlled formation
  */
  ```
- `transaction ()`
  ```
  /*
     Default constructor
  */
  ```
- `transaction (string action, int customerIDnumber, strong typeOfMedia, string movieTitle)`
  ```
  /*
     Constructor used to generate transactions inside the inventory class
  */
  ```

HashTable Class

Private Data Members:

- `int size;` `//Maintains the size of the hash table`
- `hashTableEntry** table;` `// Maintains the address of the next entry in the hash table`

Public Functions:

- `hashTable (int size)`
  ```
  /*
     Constructor taking an int as a perimeter to set the size
  */
  ```
- `int hashFunction(string key)`
  ```
  /*
     Function responsible for creating and returning a hash code that will represent the index of where the movie name will be placed in the hashTable
  */
  ```
- `void insert (string key, int value)`
  ```
  /*
     Responsible for inserting the value at the key index location
  */
  ```

HashTableEntry Class

Private Data Members:

- `string key;` `//maintains the key`
- `int value;` `//maintains the value`
- `hashTableEntry* next;` `//maintains the address to the next entry in the hashTable.`

Public Functions:

- `hashTableEntry (string key, int value, hashTableEntry* next)`
  ```
  /*
     Constructor that creates the hashTableEntry object
  */
  ```