**Team:**
Satine Paronyan
Dan Gallo
Wilbert Sible

# Automated Movie Rental Tracking System

## Overview

***Problem Statement:***

Our team needs to create an automated inventory tracking system for a movie rental store. The movies need to be inventoried and sorted specific to their by genre type. Movie inventories are updated based on specific customer commands for borrowing and returning. The customers' movie histories are tracked so the customer can check their rental history.

**Design Description:**

Our team plans to create an inheritance structure based on the movie genre types with the parent class Movie and child classes Comedy, Drama, and Classics. The genre children overwrite Movie's comparator functions for genre-specific sorting.

The Inventory class maintains three genre-specific tree sets (Comedy, Drama, and Classics) to sort the movies. The Inventory class utilizes the genre-specific class' comparators to sort its corresponding tree set when adding the new inventory to the inventory database.

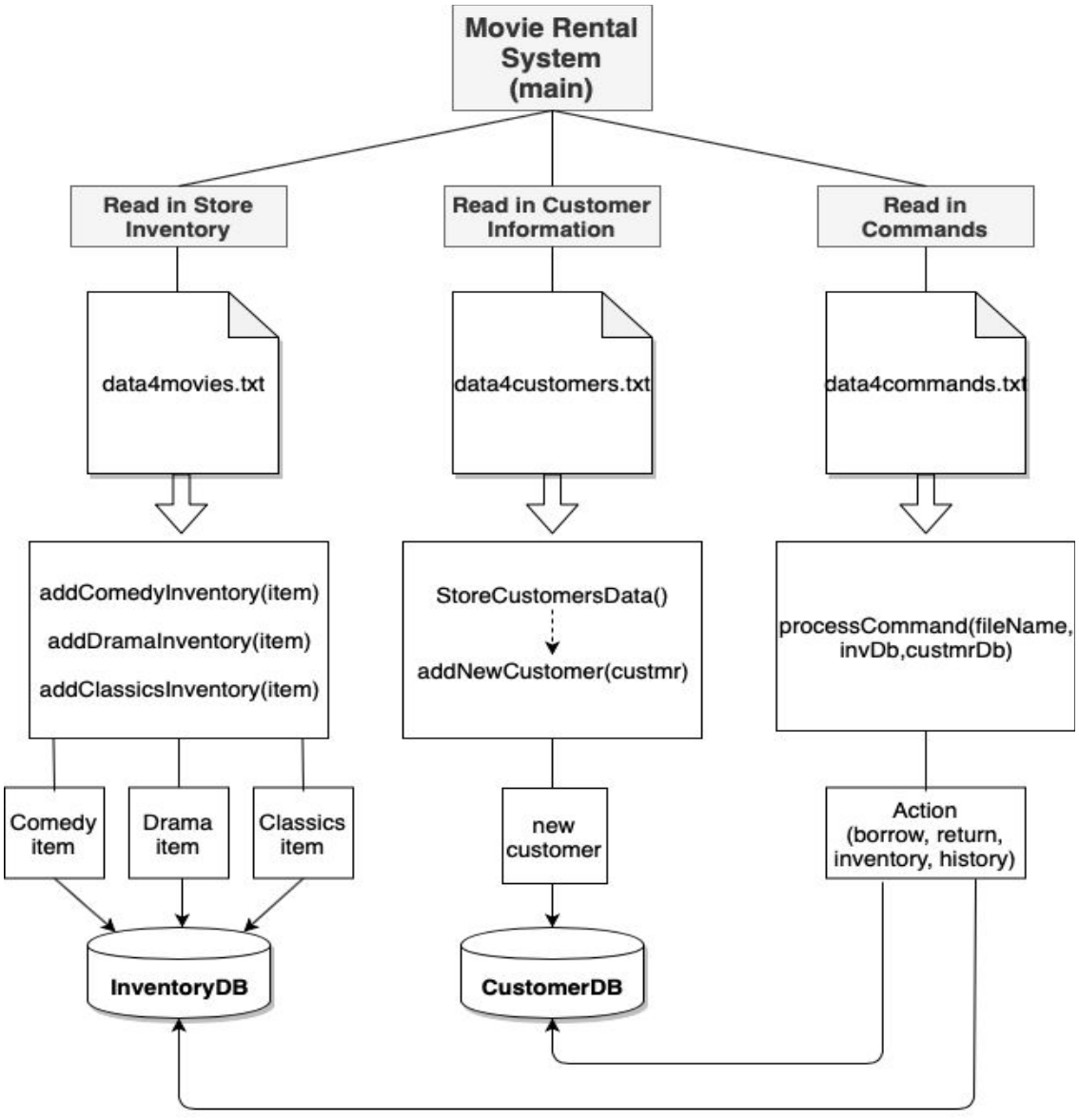
The Customer database maintains a customer hash table. The customer hash table pushes Customer objects to linked lists contained within the hash table buckets. The Customer objects contain the specific customer's rental history. The unique customer id is used to determine the where to insert that specific customer in the hashtable.

The hash table is an array of Customer pointers, we also have a data member size, which is used to initialize the size of the array, grow the array and to calculate the hash code of the new Customer object to be inserted into the hashtable. For example, if the id number is 1112, then we determine into which bucket to put that Customer object by using the value of this expression: 1112 % size of the array. The underlying structure of the hashtable will grow as soon as it adds the number of elements corresponding to size's value. Each time the

hashtable grows it rehashes all the elements in it and puts them into the new location in a newly grown hashtable.

The Action class is used to read in the commands from a data4commands.txt. This class contains a public method processCommand, which accepts three parameters: the file name to be opened and read in, the reference to the InventoryDB database and the reference to the CustomersDB database. Each line from the file is processed and if there is any unrecognized command then the user is notified and that command (line) is discarded.

All operation of the program is performed by the `main function` that connects everything together. The main first reads all the inventory to the InventoryDB, then reads all the customers into CustomersDB, then reads in the command and updates the inventory and customer databases.



**Input:**

The input will come from three input files which contain the customers' information, the movie descriptions, and the customers' commands.

The customers' information will include their unique customer ID and their full name as seen below:

```
1111 Mouse Mickey
1000 Mouse Minnie
```

The movie descriptions contain the genre, the inventory quantity, the director, the movie title, and the release date. Classic movies specifically also contain the major actor between the title and release date. Example below:

```
F, 10, Nora Ephron, You've Got Mail, 1998
D, 10, Steven Spielberg, Schindler's List, 1993
C, 10, George Cukor, Holiday, Katherine Hepburn 9 1938
```

The customer commands are comprised of the requested action, customer ID, media type, movie genre, and movie information as seen below:

```
B 1234 D F Pirates of the Caribbean, 2003
R 1234 D C 9 1938 Katherine Hepburn
B 1234 D D Steven Spielberg, Schindler's List,
```

**Output:**

The output for inventory "I" command will be all Comedy movies, then all Drama movies, and finally all Classic movies.

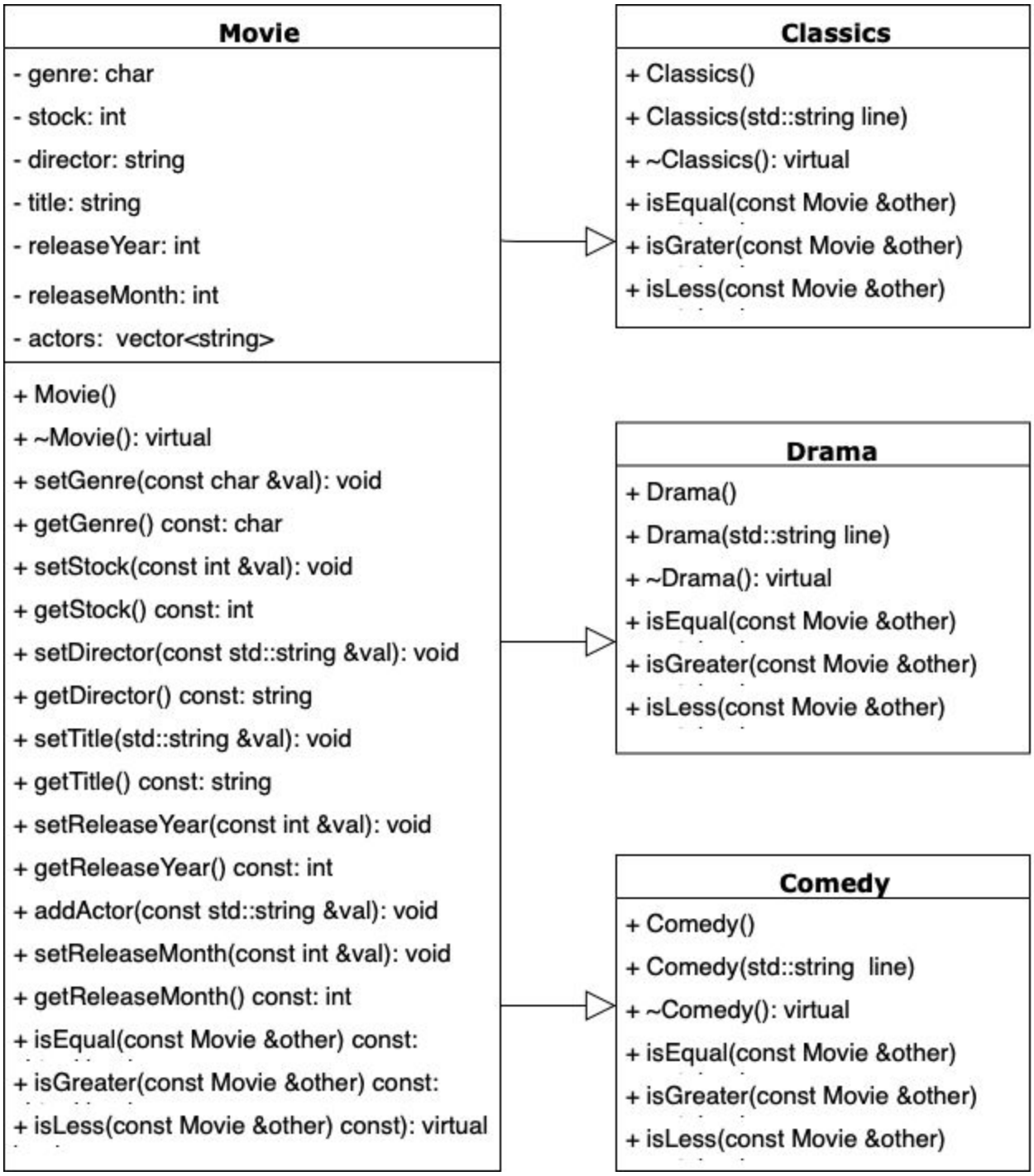
The "H" command will show a customer's DVD transactions in chronological order and give the current borrowed/returned status.

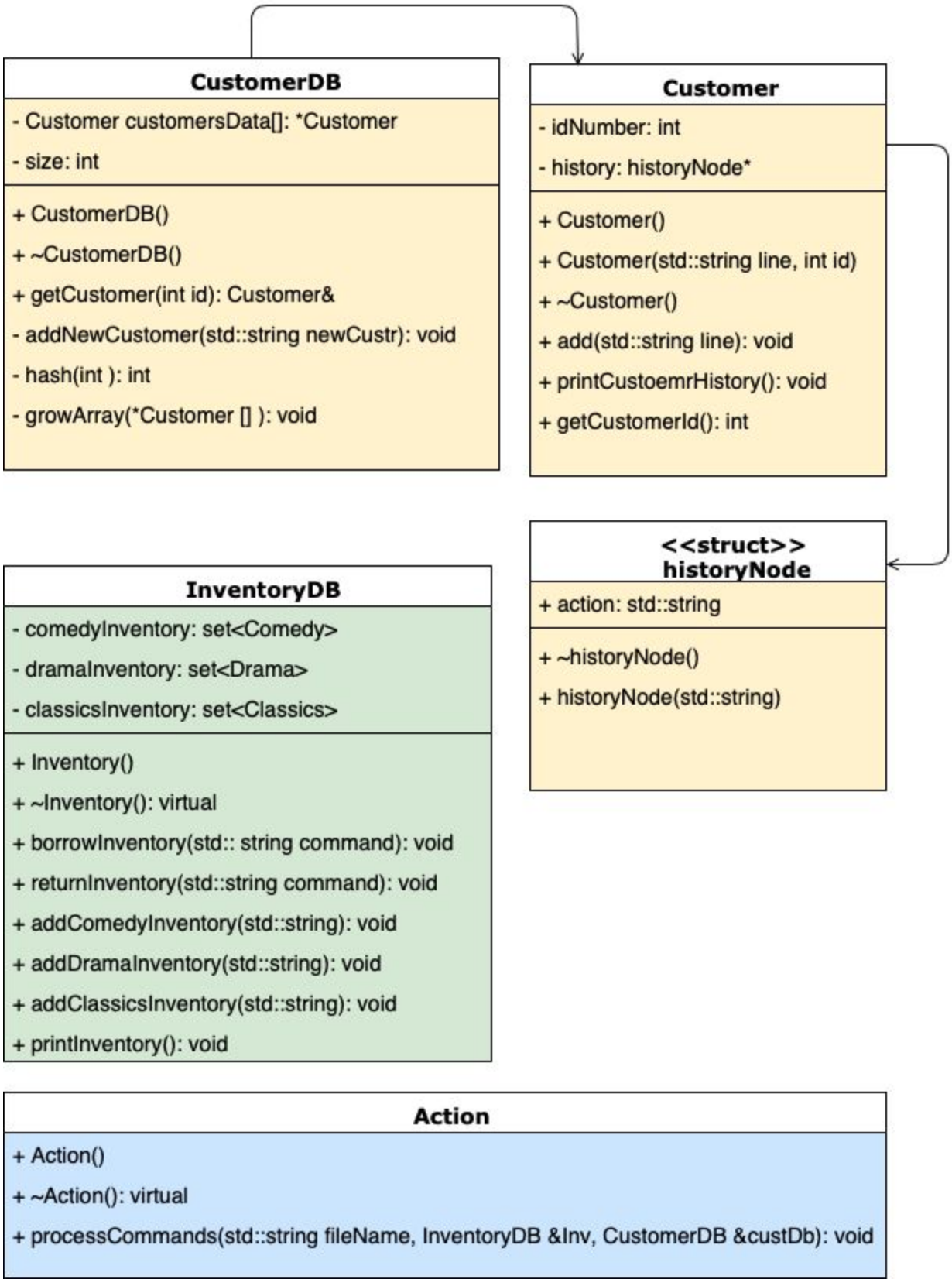
**Error Handling:**

The movie input file can potentially have invalid genre codes such as 'Z'. Invalid entries need to be removed and users told of the issue.

The command file may have invalid action codes, invalid video codes, incorrect customer IDs, and invalid movies. Invalid entries need to be removed and users told of the issue.

# Class Diagrams




CustomerDB
- Customer customersData[]: *Customer
- size: int
+ CustomerDB()
+ ~CustomerDB()
+ getCustomer(int id): Customer&
- addNewCustomer(std::string newCustr): void
- hash(int ): int
- growArray(*Customer [] ): void
Customer
- idNumber: int
- history: historyNode*
+ Customer()
+ Customer(std::string line, int id)
+ ~Customer()
+ add(std::string line): void
+ printCustoemrHistory(): void
+ getCustomerId(): int
<<struct>>
historyNode
+ action: std::string
+ ~historyNode()
+ historyNode(std::string)
InventoryDB
- comedyInventory: set<Comedy>
- dramaInventory: set<Drama>
- classicsInventory: set<Classics>
+ Inventory()
+ ~Inventory(): virtual
+ borrowInventory(std:: string command): void
+ returnInventory(std::string command): void
+ addComedyInventory(std::string): void
+ addDramaInventory(std::string): void
+ addClassicsInventory(std::string): void
+ printInventory(): void
Action
+ Action()
+ ~Action(): virtual
+ processCommands(std::string fileName, InventoryDB &Inv, CustomerDB &custDb): void

// High-level pseudocode for: processCommands in Action class

```
void Action::processCommands(std::string fileName, InventoryDB &Inv, CustomerDB &custDb)
{
   fstream file(fileName);
   if (file.fail())
   {
      cerr << "Can't find commands file: " << fileName << endl;
      return;
   }

   char actionType;
   std::string command;
   std::string idNum = 0;

   while (!file.eof())
   {
      file >> actionType;          // read first char (action type) in command line
      getline(file, command);          //read in the rest of the command line after action type
      idNum = command.substr(0, command.find(' '));        // will assign the characters until finds space

      switch (actionType)
      {
         case 'I':
            Inv.printInventory();
         case 'H':
            custDb.printCustomerHistory(std::stoi(idNum));  //stoi() converts str to int
         case 'B':
            //NOTE: after first char (action type) was extracted from line the rest
            // of the line will be past into borrowInventory and returnInventory
            Inv.borrowInventory(command);
         case 'R':
           Inv.returnInventory(command);
         default:
            //action type is incorrect
            cerr << "Unknown action type: " << actionType  << " " << command << endl;
      }
   }
} //end of processCommands
```

```
// High-level pseudocode for the program's main
int main() {

    // ---------------------- read inventory into store data base  ----------------------
    std::string fName = "data4movies.txt";
    fstream moviesFile(fName);
    if (moviesFile.fail()) {
        cerr << "Can't find movie file: " << fName << endl;
        return 1;
    }
    InventoryDB storeInventoryDB;
    std::string movieItem;   // needed for reading movie items line by line
    std::string movieGenre;  //will hold first elements of the string

    while (!moviesFile.eof()) {
        getline(moviesFile, movieItem);
        movieGenre = movieItem.substr(0, movieItem.find(' ')); // needed for switch

        switch (movieItem.at(0)) {
            case 'F':
                storeInventoryDB.addComedyInventory(movieItem);
            case 'D':
                storeInventoryDB.addDramaInventory(movieItem);
            case 'C':
                storeInventoryDB.addClassicsInventory(movieItem);
            default:
                //movie type is incorrect
                cerr << "Unknown movie genre: " << movieItem << endl;
        }
    }

    // ---------------------- read customers into the  customers data base  ----------------------
    std::string cFileName = "data4customers.txt";
    fstream customersFile(cFileName);
    if (customersFile.fail()) {
        cerr << "Can't find customers file: " << cFileName << endl;
        return 1;
    }

    CustomerDB storeCustomersData;
    std::string newCustomer;
    while (!customersFile.eof()) {
        getline(customersFile, newCustomer);
        storeCustomersData.addNewCustomer(newCustomer);
    }

    // -------------------------- execute commands from file  --------------------------
    std::string commandsFileName = "data4commands.txt";
    Action action;
    action.processCommands(commandsFileName, storeInventoryDB, storeCustomersData);

    return 0;  //everything executed successfully
}
```

# Class Description

Movie Inheritance Classes:

| Movie | |
|---|---|
| Data Members | |
| `char genre` | Movie genre data type: represented as ‘F’ for comedy, ‘D’ for drama, ‘C’ for classics. |
| `int stock` | Number of movies in stock at rental store. |
| `string director` | Film director’s full name. |
| `int releaseYear` | The release year of the film. |
| `int releaseMonth` | The release month of the film. |
| `vector<string> actors` | The main actors in the film. |
| Constructor/ Destructor | |
| `Movie()` | The default constructor initializes the data fields prior to genre specification and construction. |
| `virtual ~Movie()` | Destroys Movie object. |
| Accessors/ Mutators | |
| `void setGenre(const char &val)` | Sets the Movie instance’s genre character. |
| `char getGenre() const` | Returns the Movie instance’s genre character. |
| `void setStock(const int &val)` | Sets the Movie instance’s current stock quantity. |
| `int getStock() const` | Returns the Movie instance’s current stock quantity. |
| `void setDirector(const std::string &val)` | Sets the Movie instance director’s name. |
| `string getDirector() const` | Returns the Movie instance director’s name. |

| | |
|---|---|
| `void setTitle(std::string &val)` | Sets the Movie instance's movie title. |
| `string getTitle() const` | Returns the Movie instance's movie title. |
| `void setReleaseYear(const int &val)` | Sets the Movie instance's release year. |
| `int getReleaseYear() const` | Returns the Movie instance's release year. |
| `void setReleaseMonth(const int &val)` | Sets the Movie instance's release month. |
| `int getReleaseMonth() const` | Returns the Movie instance's release month. |
| Comparator Functions | |
| `virtual bool isEqual(const Movie &other) const` | Compares two Movie instances for equality and returns the bool true/false result. |
| `virtual bool isGreater(const Movie &other) const` | Compares if a Movie instance is greater than the other Movie instance and returns the bool true/false result. |
| `virtual bool isLess(const Movie &other) const` | Compares if a Movie instance is less than the other Movie instance and returns the bool true/false result. |
| Classics | |
| Constructors / Destructor | |
| `Classics()` | The default constructor initializes the data fields. |
| `Classics(std::string line)` | The constructor takes the string parameter and parses it into fields to set the relevant Movie data members. |
| `virtual ~Classics()` | Destroys Classics object. |
| Comparator Functions | |
| `bool isEqual(const Movie &other) const` | Compares two Classics instances for equality based on Release date then Major actor. Returns the bool true/false result. |
| `bool isGreater(const Movie &other) const` | Compares if a Classic instance is greater than the other Classic instance based on Release date then Major actor. Returns |

| | |
|---|---|
| | the bool true/false result. |
| `bool isLess(const Movie &other) const` | Compares if a Classic instance is less than the other Classic instance based on Release date then Major actor. Returns the bool true/false result. |
| **Drama** | |
| **Constructors / Destructor** | |
| `Drama()` | The default constructor initializes the data fields. |
| `Drama(std::string line)` | The constructor takes the string parameter and parses it into fields to set the relevant Movie data members. |
| `virtual ~Drama()` | Destructor destroys |
| **Comparator Functions** | |
| `bool isEqual(const Movie &other) const` | Compares two Drama instances for equality based on Director then Title. Returns the bool true/false result. |
| `bool isGreater(const Movie &other) const` | Compares if a Drama instance is greater than the other Drama instance based on Director then Title. Returns the bool true/false result. |
| `bool isLess(const Movie &other) const` | Compares if a Drama instance is less than the other Drama instance based on director then Title. Returns the bool true/false result. |
| **Comedy** | |
| **Constructors / Destructor** | |
| `Comedy()` | The default constructor initializes the data fields. |
| `Comedy(std::string line)` | The constructor takes the string parameter and parses it into fields to set the relevant Movie data members. |
| `virtual ~Comedy()` | Destructor destroys |
| **Comparator Functions** | |

| | |
|---|---|
| `bool isEqual(const Movie &other) const` | Compares two Comedy instances for equality based on Title then Year. Returns the bool true/false result. |
| `bool isGreater(const Movie &other) const` | Compares if a Comedy instance is greater than the other Comedy instance based on Title then Year. Returns the bool true/false result. |
| `bool isLess(const Movie &other) const` | Compares if a Comedy instance is less than the other instance based on Title then Year. Returns the bool true/false result. |

Customer Database Classes:

| CustomerDB | |
|---|---|
| **Data Members** | |
| `Customer* customersData[]` | Array of Customer object pointers for hash table. |
| `int size` | Size of the hash table array. |
| **Constructors / Destructor** | |
| `CustomerDB()` | The default constructor that initializes the hash table for the Customers' database. |
| `virtual ~CustomerDB()` | Destructs the hash table by destroying the Customer object pointers. |
| **Accessors/Mutators** | |
| `Customer& getCustomer(int id)` | Get specific Customer's information object. |
| `void addNewCustomer(string newCustr )` | Add a new Customer's information object to the hash table. |
| **Other Functions** | |
| `int hash(int )` | Hash function that converts a new Customer's Id to a hash index for entry. |
| `void growArray(*Customer [])` | Increase size of hash array when current array is full. |

| Customer | |
|---|---|
| Data Members | |
| `int idNumber` | Customer's unique Id number. |
| `historyNode* history` | historyNode of customer's movie transactions. |
| Constructors / Destructor | |
| `Customer()` | The default constructor initializes data members. |
| `Customer(std::string line, int id)` | The constructor parses the parameter string for customer transactions and stores the customer's unique Id. |
| `virtual ~Customer()` | Destroys the Customer pointer object. |
| Accessors/Mutators | |
| `int getCustomer()` | Returns the Customer's unique Id. |
| `void add(std::string line)` | Adds a new Customer transaction from the string parameter. |
| Input/Output Functions | |
| `void printCustomerHistory()` | Prints all Customer transactions from their historyNode. |
| <<struct>> historyNode | |
| Data Members | |
| `std::string action` | Customer's previous transaction records. |
| Constructor / Destructor | |
| `historyNode(std::string line)` | Constructs a Customer's historyNode from their first transaction (string parameter). |
| `virtual ~historyNode()` | Destroys historyNode pointer. |

Inventory Database Class:

| InventoryDB | |
|---|---|
| **Data Members** | |
| `set<Comedy> comedyInventory` | Comedy treeSet containing all the comedies in the store. |
| `set<Drama> dramaInventory` | Drama treeSet containing all the dramas in the store. |
| `set<Classics> classicsInventory` | Classics treeSet containing all the classics in the store. |
| **Constructor / Destructor** | |
| `Inventory()` | The default constructor initializes the three genre treeSets. |
| `virtual ~Inventory()` | Destroys the treeSets' pointers. |
| **Mutator Functions** | |
| `void borrowInventory(std::string command)` | Decreases inventory quantity by one for specified movie in corresponding genre treeSet based on parsed string parameter. |
| `void returnInventory(std::string command)` | Increases inventory quantity by one for specified movie in corresponding genre treeSet based on parsed string parameter. |
| `void addComedyInventory(std::string)` | Adds and sorts a new comedy movie to its corresponding genre treeSet by parsing through the string parameter to determine the movie genre and information. |
| `void addDramaInventroy(std::string)` | Adds and sorts a new drama movie to its corresponding genre treeSet by parsing through the string parameter to determine the movie genre and information. |
| `void addClassicsInventory(std::string)` | Adds and sorts a new drama movie to its corresponding genre treeSet by parsing through the string parameter to determine the movie genre and information. |
| **Output Function** | |
| `void printInventory()` | Prints entire movie catalogue and inventory through inorder traversal of the Comedy, Drama, and Classics treeSets in that order. |

Atcion Class:

| Action | |
|---|---|
| Constructor / Destructor | |
| `Action()` | The default constructor initializes the Action objects. |
| `virtual ~Action()` | Destroys Action object. |
| | |
| `void processCommands(std::string fileName, InventoryDB &Inv, CustormerDB &custDb)` | Receives the string, which represents certain command to update information in the Inventory database and customer's database. If there is an unrecognized command the method will notify the user and discard that command. |