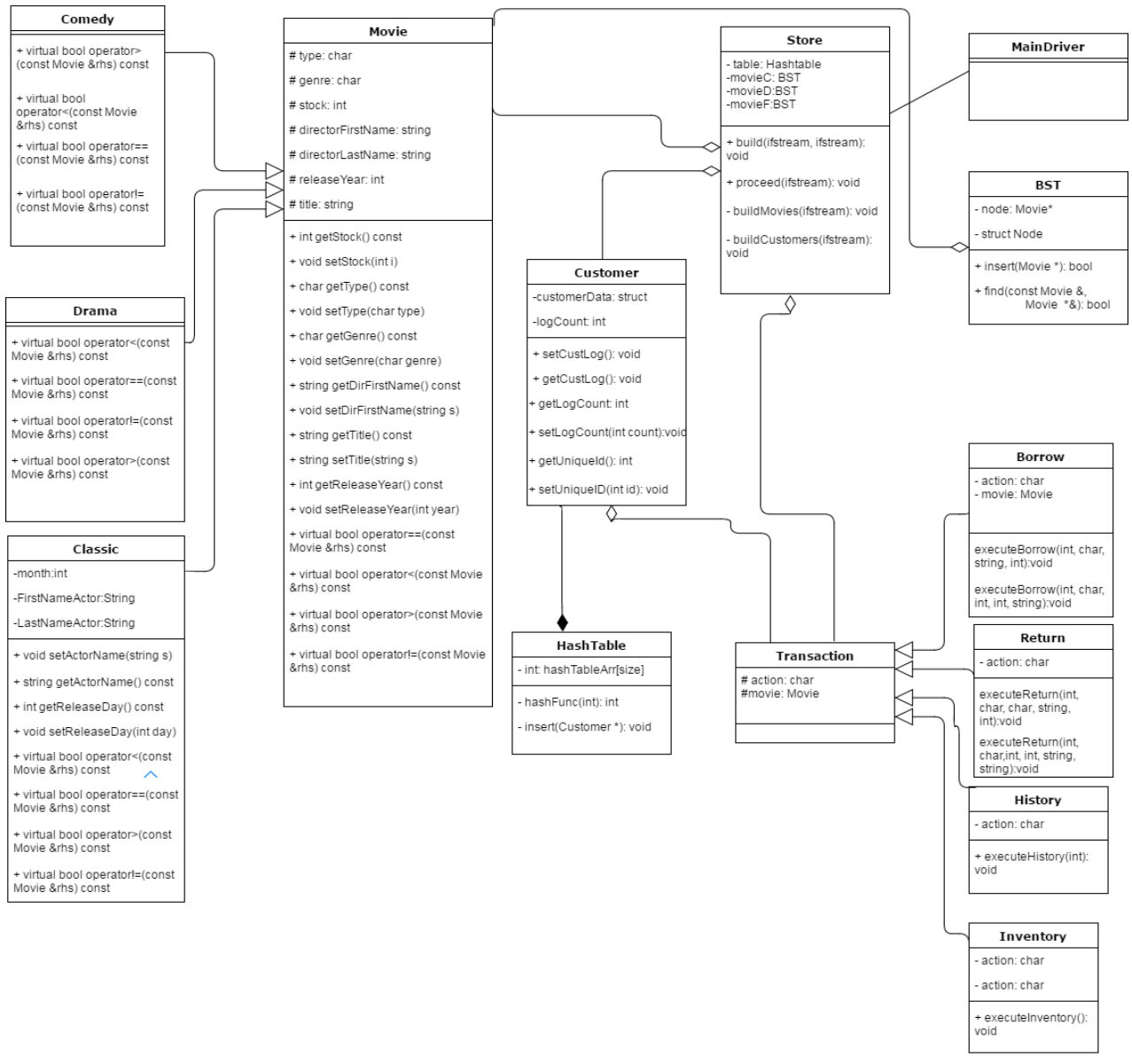
UW Bothell CSS 343:
Spring 2017
Program 4 Design
Group 8: Artiom Voronin, Natnael Tezera, Bao Tran, Jason Gautama, Jin Kim
Overview:
We have a main class called Store, which will handle all the interactions between the customer and the store by executing the commands file. Store has three BST (for efficiency) that represent three different movie genres. Store also has customers, which will be stored in a hashtable. Store interacts with class Customer which contains data members of name, ID, and history. Movie which is the base class interacts with customer. Class comedy, drama, and classic all inherit data members specified in the Movie class. Additionally, Classic class will extend on Movie class with supplementary data members such as major actor and release date. Transaction class is the base class that holds all the commands that the program will execute. Thus, Inventory, History, Borrow, and Return classes are the children of the Transaction class.

Figure 1. UML Diagram for Program 4



Description of main:
Reads text file “data4movies.txt” and initializes the content of inventory, reads another file “data4customers.txt” to get customer list, and reads “data4commands.txt” for arbitrary sequences of commands.

Main.cpp - Creates the store and executes the program

Example of main:

```
int main() {
	// reading the movies data
	ifstream movies("data4movies.txt");
	if (!movies) {
		cout << "File could not be opened." << endl;
		return 1;
	}
	// reading customers
	ifstream customers("data4customers.txt");
	if (!customers) {
		cout << "File could not be opened." << endl;
		return 1;
	}
	// reading the commands
	ifstream commands("data4commands.txt");
	if (!commands) {
		cout << "File could not be opened." << endl;
		return 1;
	}
	// creating store
	Store movieRentalStore = new Store();

	// build inventory and customer base for the store
	// movies will be stored in an array of BST Trees
	// customers are stored as hash table
	movieRentalStore.build(movies, customers);

	// execute the program by calling the commands
	movieRentalStore.proceed(commands);

	return 0;
}
```

Description of Classes:

1. **Class name: Movie**

Movie class is a parent class for Comedy, Drama, and Classic classes. Movie has protected data members: stock, directorFirstname, directorLastname, title, releaseYear, and type.

```
//-----------------------------------------------Movie-----------------------------------------------
// Purpose: This is the header file for movie.cpp
//  -- base class for comedy, drama and classic
// -- assumes comedy, drama and classic classes extend from this class
//---------------------------------------------------------------------------------------------------
#pragma once
#include <iostream>
using namespace std;

class Movie{
        // overloading operator
         friend ostream& operator<<(ostream &, const Movie &);
Public:
         //---------------------------------------Movie()-----------------------------------------
         // Default constructor
         // precondition: None
         // postconditon: a default movie class is constructed
         //---------------------------------------------------------------------------------------
         Movie();

        //---------------------------------Movie()-----------------------------------------------
         // Constructor with parameters
         // precondition: the arguments in the parameter are valid
         // postconditon: a movie with valid parametersd is initialized
         //---------------------------------------------------------------------------------------
         Movie(int stock, string firstName, string lastName, string title, int release, char
                                                                    type, char genre);

         //-------------------------------Destructor--------------------------------------------
         // Destructor: destruct the movie class
         // precondition: a constructed movie existed
         // postconditon: NONE
         //---------------------------------------------------------------------------------------
         ~Movie() { };
```

```
//---------------------------------getStock-------------------------------------------------------------
        // getStock: counts and returns the number of all available movies in the store
        // precondition: none
        // postconditon: the number of movies in the store returned
        //-------------------------------------------------------------------------------------------------
        int getStock() const;

        //---------------------------------setStock-------------------------------------------------------
        // setStock: sets the number of movies in the store
        // precondition: s is a valid integer number
        // postconditon: the number of movies in the store set
        //-------------------------------------------------------------------------------------------------
        void setStock(int i);

        //---------------------------------getType--------------------------------------------------------
        // getType:  return the type of a movie
        // precondition: none
        // postcondition: the type of a movie is returned
        //-------------------------------------------------------------------------------------------------
        char getType() const;

        //-----------------------------------setType----------------------------------------------------
        // setType: sets the type of a movie
        // precondition: c is a valid char character
        // postconditon: the type of a movie is store set
        //-------------------------------------------------------------------------------------------------
        void setType(char type);

        //---------------------------------getGenre-------------------------------------------------------
-
        // getGenre:  return the genre of a movie
        // precondition: none
        // postcondition: the genre of a movie is returned
        //-------------------------------------------------------------------------------------------------
        char getGenre() const;

        //---------------------------------setGenre-------------------------------------------------------
        // setGenre: sets the genre of a movie
        // precondition: genre is a valid char character
        // postconditon: the type of genre is store set
        //-------------------------------------------------------------------------------------------------
        void setGenre(char genre);
```

```
 //--------------------------------getDirFirstName-----------------------------------------------
 // getDirFirstName:  returns a movie's director first name
 // precondition: None
 // postconditon: the movies director first name returned
 //-----------------------------------------------------------------------------------------------
 string getDirFirstName() const;

//--------------------------------setDirFirstName-----------------------------------------------
// setDirFirstName: sets the director's first name
// precondition: s is a valid string
// postconditon: the director's first name  isset
//-----------------------------------------------------------------------------------------------
 void setDirFirstName(string s);

 //--------------------------------getDirLastName------------------------------------------------
 // getDirLastName:  returns a movie's director last name
 // precondition: None
 // postconditon: the movies director last name returned
 //-----------------------------------------------------------------------------------------------
 string getDirLastName() const;

//--------------------------------------setDirLastName-------------------------------------------
 // setDirLastName: sets the director's last name
 // precondition: s is a valid string
 // postconditon: the director's last name  is set
 //-----------------------------------------------------------------------------------------------
 void setDirLastName(string s);

 //---------------------------------------getTitle-----------------------------------------------
 // getTitle:  returns a movie title
 // precondition: None
 // postcondition: the movies title is returned
 //-----------------------------------------------------------------------------------------------
 string getTitle() const;

 //---------------------------------------setTitle-----------------------------------------------
 // setTitle: sets the movie title
 // precondition: s is a valid movie title
 // postcondition: the director's last name  is set
 //-----------------------------------------------------------------------------------------------
 void setTitle(string s);
```

```
 //----------------------------------------getReleaseYear----------------------------------------
 // getReleaseYear: returns the release year
 // precondition: None
 // postcondition: the release year is returned
//-----------------------------------------------------------------------------------------------
 int getReleaseYear() const;


//----------------------------------------setReleaseYear----------------------------------------
 // setReleaseYear: sets the release year of the move
 // precondition: year is a valid integer number > 1900
 // postcondition: the release year of a movie is returned
 //-----------------------------------------------------------------------------------------------
 void setReleaseYear(int year);

 // overload operators, for comparing movies in order to store the movies

//----------------------------------------operator<----------------------------------------------
 // operator<    : operator that compares if lhs is less than rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this < rhs; returns false if *this > rhs
 //-----------------------------------------------------------------------------------------------
virtual bool operator<(const Movie &rhs) const;

 //----------------------------------------operator>----------------------------------------------
 // operator>: checks if the lhs is greater than the rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this >  rhs; returns false if *this < rhs
 //-----------------------------------------------------------------------------------------------
virtual bool operator>(const Movie &rhs) const;

//----------------------------------------operator==---------------------------------------------
 // operator==: compares two movies to each other
 // precondition: movies exist
 // postcondition: returns true if the movies are identical, false otherwise
 //-----------------------------------------------------------------------------------------------
virtual bool operator==(const Movie &rhs) const;

//----------------------------------------operator!=---------------------------------------------
// operator!=: not equal
// Precondition: none
// Postcondition: return true if not equal rhs, else false
//-----------------------------------------------------------------------------------------------
```

```
        virtual bool operator!=(const Movie &rhs) const;

protected:
        char type;                  // Holds type of Media
        char genre;                 // movie genre
        int stock;                  // number of movies available
        string directorFirstName;   // Director's first name
        string directorLastName;    // Director's last name
        string title;               // Movie title
        int releaseYear;            // Movie release year
}
```

2. Class name: Comedy

This class is a children of Movie that inherited the Movie functions, including Stock, Director, Title, and Year. Handles actions related to the comedy class

```
#pragma once
#include "movie.h"
// ------------------------------ comedy.h-----------------------------------------------------------------
// Purpose: This file contains Comedy class function declaration notes:
//- Comedy Class inherits the attributes from Movie Class
//-------------------------------------------------------------------------------------------------------------

class Comedy : public Movie
{
public:
        //-------------------------------------customer()-----------------------------------------------------
        // Customer()          : default constructor for Comedy class
        // precondition        : NONE
        // postcondition       : initialize the private value to NULL
        //-------------------------------------------------------------------------------------------------------
        Comedy();


        //-------------------------------------------------------------------------------------------------------
        // Customer()          : constructor that set the datas
        //                       for Comedy class
        // precondition        : NONE
        // postcondition       : initialize the private value to NULL
        //-------------------------------------------------------------------------------------------------------
        Comedy(int stock, string firstName, string lastName, string title, int
                                        release, char type, char genre);
        //-------------------------------------------------------------------------------------------------------
```

```
// ~Customer()          : destructor for Customer class
// precondition         : NONE
// postcondition        : NONE
//-----------------------------------------------------------------------------------------------------
virtual ~Comedy() {};

//---------------------------------------operator<-------------------------------------------------
 // operator<    : operator that compares if lhs is less than rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this < rhs; returns false if *this > rhs
 //-----------------------------------------------------------------------------------------------------
virtual bool operator<(const Movie &rhs) const;

 //---------------------------------------operator>-------------------------------------------------
 // operator>: checks if the lhs is greater than the rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this >  rhs; returns false if *this < rhs
 //-----------------------------------------------------------------------------------------------------
virtual bool operator>(const Movie &rhs) const;

//---------------------------------------operator==-------------------------------------------------
 // operator==: compares two movies to each other
 // precondition: movies exist
 // postcondition: returns true if the movies are identical, false otherwise
 //-----------------------------------------------------------------------------------------------------
virtual bool operator==(const Movie &rhs) const;

//---------------------------------------operator!=-------------------------------------------------
// operator!=: not equal
// Precondition: none
// Postcondition: return true if not equal rhs, else false
//-----------------------------------------------------------------------------------------------------
virtual bool operator!=(const Movie &rhs) const;};
```

3. Class name: Classic

Extends from the movie class, and includes additional attributes including major actor and release date. It handles actions related to classic movies

```
#pragma once
#include <iostream>
#include "movie.h"
class Classic: public Movie{
        friend ostream& operator <<(ostream &, const Classic &);
```

Public:

```
//---------------------------------Classic-----------------------------------------------------------
// Default constructor
// precondition: None
// postconditon: a default classic class is constructed
//---------------------------------------------------------------------------------------------------
Classic();

//---------------------------------Classic-----------------------------------------------------------
// Constructor
// precondition: the arguments in the parameter are valid
// postconditon: a classic movie with valid parameter is initialized
//---------------------------------------------------------------------------------------------------
Classic(int stock, string firstName, string lastName, string title, int
                                                    release, char type, char genre);

//---------------------------------Destructor--------------------------------------------------------
// Destructor: destructs the classic class
// precondition: a constructed movie existed
// postconditon: set to NULL
//---------------------------------------------------------------------------------------------------
virtual ~Classic();

//---------------------------------setActorName------------------------------------------------------
// setActorName: sets actor first name
// precondition: s is a valid string object
// postconditon: the actor's first name of a movie is store set
//---------------------------------------------------------------------------------------------------
void setActorName(string s);

//---------------------------------getTitle----------------------------------------------------------
// getTitle:  returns actor fname
// precondition: None
// postconditon: the actor name is returned
//---------------------------------------------------------------------------------------------------
string getActorName();

//---------------------------------getStock----------------------------------------------------------
// getReleaseDay: returns release date
// precondition: None
// postconditon: release date returned
//---------------------------------------------------------------------------------------------------
int getReleaseDay() const;
```

```
 //--------------------------------setReleaseDay-----------------------------------------------
 // setReleaseDay: sets the release date
 // precondition: i is a valid integer number
 // postcondition: the release date is set
 //-----------------------------------------------------------------------------------------------
 void setReleaseDay(int day);

//-----------------------------------------operator<-----------------------------------------------
 // operator<    : operator that compares if lhs is less than rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this < rhs; returns false if *this > rhs
 //-----------------------------------------------------------------------------------------------
virtual bool operator<(const Movie &rhs) const;

 //-----------------------------------------operator>-----------------------------------------------
 // operator>: checks if the lhs is greater than the rhs
 // precondition: rhs is a valid Movie type
 // postcondition: returns true if *this >  rhs; returns false if *this < rhs
 //-----------------------------------------------------------------------------------------------
virtual bool operator>(const Movie &rhs) const;

//-----------------------------------------operator==----------------------------------------------
 // operator==: compares two movies to each other
 // precondition: movies exist
 // postcondition: returns true if the movies are identical, false otherwise
 //-----------------------------------------------------------------------------------------------
virtual bool operator==(const Movie &rhs) const;

//-----------------------------------------operator!=----------------------------------------------
// operator!=: not equal
// Precondition: none
// Postcondition: return true if not equal rhs, else false
//-----------------------------------------------------------------------------------------------
virtual bool operator!=(const Movie &rhs) const;


private:
        string actortName;          // First name of actor
        int releaseDay;             // release day of movie
}
```

4. Class name: Drama

extends from the movie class and implement functions that relates with drama movies. It handles actions related to the drama class

```
#pragma once
#include <iostream>
#include “movie.h”
class Drama : public Movie
{
private:

Public:
        //---------------------------------drama-------------------------------------------------------------
         // Default constructor
         // precondition: None
         // postconditon: a default classic class is constructed
         //-------------------------------------------------------------------------------------------------
        Drama();

         //---------------------------------Classic-----------------------------------------------------------
         // Constructor
         // precondition: the arguments in the parameter are valid
         // postconditon: a classic movie with valid parameter is initialized
         //-------------------------------------------------------------------------------------------------
        Drama(int stock, string firstName, string lastName, string title, int release, char
                type, char genre);

         //---------------------------------Destructor--------------------------------------------------------
         // Destructor: destructs the classic class
         // precondition: a constructed movie existed
         // postconditon: set to NULL
         //-------------------------------------------------------------------------------------------------
        virtual ~Drama() ;

        //---------------------------------------operator<-----------------------------------
         // operator<    : operator that compares if lhs is less than rhs
         // precondition: rhs is a valid Movie type
         // postcondition: returns true if *this < rhs; returns false if *this > rhs
         //-------------------------------------------------------------------------------------------------
        virtual bool operator<(const Movie &rhs) const;

         //---------------------------------------operator>-----------------------------------
         // operator>: checks if the lhs is greater than the rhs
```

```
        // precondition: rhs is a valid Movie type
        // postcondition: returns true if *this >  rhs; returns false if *this < rhs
        //-----------------------------------------------------------------------------------------------
        virtual bool operator>(const Movie &rhs) const;

        //----------------------------------------operator==-----------------------------------------
        // operator==: compares two movies to each other
        // precondition: movies exist
        // postcondition: returns true if the movies are identical, false otherwise
        //-----------------------------------------------------------------------------------------------
        virtual bool operator==(const Movie &rhs) const;

        //----------------------------------------operator!=-----------------------------------------
        // operator!=: not equal
        // Precondition: none
        // Postcondition: return true if not equal rhs, else false
        //-----------------------------------------------------------------------------------------------
        virtual bool operator!=(const Movie &rhs) const;

};
```

5. Class name: Customer

Customer class is a class that will store the customerData struct; countHistory, which counts the number of history transactions the user has used as index; and also the History of the borrowed item.

```
#pragma once
#include "movie.h"
#include “transaction.h”
#include <string>
using namespace std;


// ----------------------------------customer.h ------------------------------------------------------------
// Purpose: This file contains customer Class function declaration.
// ---------------------------------------------------------------------------------------------------------

#define MAXSIZE 100
class  Customer {
private:

        //forward declaration
        struct CustomerData;
        int logCount = 0;                                   // count the number of history
```

```
                                                    // transaction the user has Used as
                                                    // index.
        struct CustomerData {
                int uniqueID;                       // customer's unique identification
                int uniqueKey;                      // customer's unique key for
hasTable
                Transaction log[MAXSIZE];           // user's log
                string firstName;                   // customer's first name
                string lastName;                    // customer's last name
        };
public:

        //-------------------------------------------------------------------------------------------------
        // Customer()          : default constructor for Customer class
        // precondition        : NONE
        // postcondition       : initialize the private value to NULL
        //-------------------------------------------------------------------------------------------------
        Customer();

        //-------------------------------------------------------------------------------------------------
        // ~Customer()         : destructor for Customer class
        // precondition        : NONE
        // postcondition       : delete the array of Transaction
        //-------------------------------------------------------------------------------------------------
        ~Customer();

        //-------------------------------------------------------------------------------------------------
        // Customer()          : constructor that set uniqueID and customer's name
        //                       for customer
        // precondition        : uniqueID is four digit number, firstName and lastName
        //                       should be a valid string
        // postcondition       : initialize the private values with the passed in values
        //-------------------------------------------------------------------------------------------------
        Customer(int uniqueID, string firstName, string lastName);

        //-------------------------------------------------------------------------------------------------
        // getCustLog()        : accessor to access customer's history by calling
        //                       hash table class
        // precondition        : NONE
        // postcondition       : print the customer’s history
        //-------------------------------------------------------------------------------------------------
        void getCustLog();
```

```
    //-------------------------------------------------------------------------------------------
    // setCustLog()         : mutator to set customer's history
    // precondition         : t is a valid Transaction type
    // postcondition        : customer's history is set
    //-------------------------------------------------------------------------------------------
    void setCustLog(Transaction &t);

    //-------------------------------------------------------------------------------------------
    // getLogCount()        : accessor to access the number of borrow/ return
    //                        transactions
    //                      history the customer has
    // precondition         : NONE
    // postcondition        : return customer's history count
    //-------------------------------------------------------------------------------------------
    int getLogCount() const;

    //-------------------------------------------------------------------------------------------
    // setLogCount(): mutator to set the number of borrow/ return
    //                                          history the customer has
    // precondition         : count > 0
    // postcondition        : customer's history count is set
    //-------------------------------------------------------------------------------------------
    void setLogCount(int count);

    //-------------------------------------------------------------------------------------------
    // getUniqueID()        : accessor to access customer's unique ID
    // precondition         : NONE
    // postcondition        : return customer's unique ID
    //-------------------------------------------------------------------------------------------
    int getUniqueID() const;



    //-------------------------------------------------------------------------------------------
    // setUniqueID()        : mutator to set customer's unique ID
    // precondition         : uniqueID is a valid int, 4 digits
    // postcondition        : customer's unique ID are set
    //-------------------------------------------------------------------------------------------
    void setUniqueID(int uniqueID);

}; // end of class Customer
```

6. Class name: Store

Store is a wrapper class for customers and 3 BST of movies. It reads the three data files(data4Customers.txt, data4Movies.txt, data4Commands.txt) and builds a binary search tree of drama, comedy and classic movies, and fill a hashTable with the customer information.

```
// ---------------------------------- store.h ----------------------------------------------------------------
// Purpose: This file contains Store class function declaration
// -----------------------------------------------------------------------------------------------------------
Class Store{
Public:
    //-------------------------------------------------------------------------------------------------------
    // Store()              : default constructor for Customer class
    // precondition         : NONE
    // postcondition        : builds store object
    //-------------------------------------------------------------------------------------------------------
    Store();

    //-------------------------------------------------------------------------------------------------------
    // ~Store()             : destructor for Store class
    // precondition         : NONE
    // postcondition        : store object is destroyed
    //-------------------------------------------------------------------------------------------------------
    ~Store();

    //-------------------------------------------------------------------------------------------------------
    // build()              : function that takes in movies and customers
    //                        as a parameter
    // precondition         : movies and customers have valid data
    // postcondition        : movies and customers are built
    //-------------------------------------------------------------------------------------------------------
    void build(ifstream &movies, ifstream &customers)
    {
      // call buildMovies(movies)
      // call buildCustomers(customers);
    }


    //-------------------------------------------------------------------------------------------------------
    // proceed              : reads the commands and executes them
    // precondition         : commands are valid format
    // postcondition        : executes the commands specified by the file
    //-------------------------------------------------------------------------------------------------------
    void proceed(ifstream &commands)
        {
```

```
        char ch = ‘’;
        commands >> ch;
        Switch (ch) {
        Case ‘I’:
                call inventory transaction
                Break;
        Case ‘H’:
                call history transaction
                Break;
        Case ‘B’:
                call borrow transaction
                Break;
        Case ‘R’:
                call return transaction
                Break;
        }

private:
    //-----------------------------------------------------------------------------------------------
    // buildMovies(ifstream &movie);
    // precondition: movie file exists
    // postcondition: all movies are processed and stored in their BST
    //-----------------------------------------------------------------------------------------------
    buildMovies(ifstream &movie);
        while(data4movies.txt is not empty, or end of the file){
            Read the genre of the movie
            If the genre is comedy, Insert it in the comedy bst
            If the genre is drama, insert the movie in the drama bst
            If the genre is classic, insert the movie in the classic
            Skip to the next line
        }




    //-----------------------------------------------------------------------------------------------
    // buildCustomers(ifstream &customer)
    // precondition: customer file exists
    // postcondition: a hashtable is built and store customers by their key, which
    // come from their unique ID
    //-----------------------------------------------------------------------------------------------
    buildCustomers(ifstream &customer)
    {
```

```
                HashTable ht;
                while(data4Customer.txt is not empty, or end of the file){
                        Read the customerID, first name and last name
                        Create new Customer object called cust
                        ht.insert(cust);
                        Skip to the next line
                }
        }
        HashTable table;
        BST *movieC;
        BST *movieD;
        BST *movieF;
};
```

7. Class name: Transaction -- Transaction class is parent class of borrow and return class

```
//-------------------------Transaction-----------------------------------------------------------
// This class contain a declaration for Transaction class
//---------------------------------------------------------------------------------------------------------
#include “Movie.h”

Using namespace std;
Class Transaction {
Protected:
        Char action = ‘’;
        Movie movie;
Public:
        //---------------------Default Constructor-------------------------------------------------
        //precondition  : NONE
        //Postcondition : A new transaction object is created
        //---------------------------------------------------------------------------------------------
        Transaction();

        //---------------------Constructor-----------------------------------
        //precondition: A Movie object is passed in
        //Postcondition : A new transaction object is created
        //Set movie to be movie in parameter
        //-------------------------------------------------------------------
        Transaction(Movie mov);


        //--------------------Destructor-----------------------------------------------------------
        // Precondition:
```

```
        // Postcondition: A new transaction object is destroyed
        //-----------------------------------------------------------------------------------------------
        ~ virtual Transaction();


};//end class transaction
```

8. Class name: Borrow -- Borrow class is borrow transaction, which will be used when a customer borrow a movie from store. Action character: B. This class will interact with Customer and Store classes. Write a new transaction to Customer's history and decrease one in stock of Store class.

```
Class Borrow: public Transaction{
Private:
        Char action = 'B';              //overloaded from transaction
        Movie movie = NULL;             //inherited from transaction
Public:

        //--------------------Default Constructor-------------------------------------------------
        // Borrow();
        // Precondition:
        // Postcondition: A borrow transaction is created
        //-----------------------------------------------------------------------------------------------
        Borrow();

        //------------------Constructor(Movie)---------------------------------------------------
        // Borrow(movie);
        // Precodition: A Movie object is passed in
        // Postcondition: A borrow transaction is created
        // Movie is movie from parameter
        //-----------------------------------------------------------------------------------------------
        Borrow(Movie &movie);

        //-----------------------Destructor-----------------------------------------------------
        // ~Borrow();
        // Precondition:
        // Postcondition: Borrow object is destroyed
        //-----------------------------------------------------------------------------------------------
        ~Borrow();

        //--------------------executeBorrow()-----------------------------------------------------
        // Description: execute borrow function for comedy
        // Precondition:
```

```
        // Postcondition: stock in store -1, a new transaction
        //                 is added in corresponding  customer's history
        //-------------------------------------------------------------------------------------------------
        void executeBorrowC(int customerID,
        char movieType, string title, int year);

        //--------------------executeBorrow()-------------------------------------------------------------
        // Description: execute borrow function for drama movies
        // Precondition:
        // Postcondition: stock in store -1, a new transaction
        //         is added in corresponding  csutomer's history
        //-------------------------------------------------------------------------------------------------
        void executeBorrowD(int customerID, char movieType,
        String director, string title);

        //--------------------executeBorrow()-------------------------------------------------------------
        //Description: execute borrow function for comedy
        //         and drama movies
        // Precondition:
        //Postcondition: stock in store -1, a new transaction
        //         is added in corresponding  customer's history
        //-------------------------------------------------------------------------------------------------
        void executeBorrow(int customerID, char movieType,
        Int month, int year, string actor);

};//End Borrow class
```

9. Class name: Return -- Return class is return transaction, which will be used when a customer return a movie to the store. Action character: R.

```
//-------------------------  class Return ---------------------------------
// Return is a child of the Transaction Class.
// Return affects two objects: 1. A customer
//                 Updates the history log for the customer
//                                 2. The store
//                 Updates the inventory and the status of the movie
//-------------------------------------------------------------------------

class Return : public Transaction {

private:
```

```
        char action = 'R'; // action identifier for Return transaction

    public:
        //--------------------Default Constructor----------------//
        // Return();
        // Precondition:  none
        // Postcondition: Return transaction is created
        //-------------------------------------------------------//
        Return();

        //--------------------Destructor-----------------------//
        // ~Return();
        // Precondition:  none
        // Postcondition: Return transaction is deleted
        //-------------------------------------------------------//
        ~Return();

        //--------------------executeReturn-----------------------//
        // executeReturn: this function is designed for F&D type of genres.
        // Precondition:  all parameters must exist
        // Postcondition: return is performed by updating the inventory of the store,
        //                                  movie status is updated, customer log is updated.
        //-------------------------------------------------------//
        void executeReturn(int cutomerID, char mediaType, char movieType, string
    movieName,
                                    int year); // return for F&D(Comedy&Drama) genres

        //--------------------executeReturn-----------------------//
        // executeReturn: this function is designed for C type of genre.
        // Precondition:  all parameters must exist
        // Postcondition: return is performed by updating the inventory of the store,
        //                                  movie status is updated, customer log is updated.
        //-------------------------------------------------------//
        void executeReturn(int cutomerID, char mediaType, int month, int year, string
    directorFirst, string directorLast); // return for C(classical) genre

    };
```

10. Class name: Inventory-- Inventory class is an inventory transaction, which will be used when a store would like to print out its inventory. Action character: I.

```
//-------------------------  class Inventory ---------------------------------
// Inventory is a child of the Transaction Class.
// Inventory outputs all the movies that the store carries.
```

```
//---------------------------------------------------------------------------


class Inventory : public Transaction {

private:
        char action = 'I'; // action identifier for Inventory transaction

public:
        //--------------------Default Constructor----------------//
        // Inventory();
        // Precondition:  none
        // Postcondition: Inventory transaction is created
        //-------------------------------------------------------//
        Inventory();

        //--------------------Destructor-----------------------//
        // ~Inventory();
        // Precondition:  none
        // Postcondition: Inventory transaction is deleted
        //-------------------------------------------------------//
        ~Inventory();

        //--------------------executeInventory-----------------------//
        // executeInventory: prints out the inventory of the store
        // Precondition:  store must be assigned and contain customers and inventory data
                          members
        // Postcondition: prints out the inventory
        //-------------------------------------------------------------//
        void executeInventory();

};
```

11. Class name: History -- History class is a history transaction, which will be used to output the log of the transactions for a customer. Action character: H.

```
//-------------------------  class History ---------------------------------
// History is a child of the Transaction Class.
// History outputs the log of the transactions between the store and a customer
//---------------------------------------------------------------------------


class History : public Transaction {
```

```
private:
        char action = 'H; // action identifier for History transaction

public:
        //--------------------Default Constructor----------------//
        // History();
        // Precondition:  none
        // Postcondition: History transaction is created
        //-------------------------------------------------------//
        History();

        //--------------------Destructor-----------------------//
        // ~History();
        // Precondition:  none
        // Postcondition: History transaction is deleted
        //-------------------------------------------------------//
        ~History();

        //--------------------executeHistory-----------------------//
        // executeHisotry: prints out the log of transactions for the customer
        // Precondition:  store must be assigned and contain customers and inventory data
        //          members customer must have a log member with transactions stored in it
        // Postcondition: prints out the log of a customer
        //-------------------------------------------------------------//
        void executeHistory(int customerID);

};
```

12. Class name: HashTable -- HashTable is implemented to handle storing necessary storage in this project. It will implement hashing function double probing.

```
        //-----------------------------------------------------------------------------
        // Authors: Group 8
        // CSS 343 Program 4
        // Created on: 5/17/ 2017
        // last Modified:
        //-----------------------------------------------------------------------------
        // Purpose: This is the header file for hashtable.cpp.
        //  -- holds customer, sort them based on their Account ID
        //-----------------------------------------------------------------------------
        #pragma once
        #include "customer.h"
```

```
#define HASHTABLESIZE 31

class HashTable{

private:
        Int hashTableArr[HASHTABLESIZE];

         //--------------------hashFunc-----------------------------------------------------------
         // hashFunc: determines where a customer go, help to insert method
         // precondition: id is a valid customer id
         // postcondition: the location where the customer need to be inserted is
         //   determined
         //-------------------------------------------------------------------------------------
         int hashFunc(int idNum)
        {
                Int key = idNum % 31;
                If (hashTableArr[key] == NULL)
                {
                        Insert the idNum;
                        Return key;
                }
                Else
                {
                        For (int i = 1; i <= HASHTABLESIZE; i++)
                        {
                         If (hashTableArr[(Key + i^2) % 31] == NULL)
                                Insert the idNum and break
                        }
                }

        Return key;

        }

public:
         //-------------------Destructor---------------------------------------
         // Destructor: destructor
         // precondition: none
         // postcondition: the hashtable is destructed
         //--------------------------------------------------------------------
         ~HashTable();

         //-------------------Constructor--------------------------------------
```

```
        // constuctor: initializes a hash table with i length
        // precondition: the parameter i is a valid integer number
        // postcondition: a length of i hash table constructed
        //-------------------------------------------------------------------
        HashTable(int i);
        //--------------------inserts-------------------------------------------
        // insert: inserts a customer in the hashtable
        // precondition: c is a valid customer object
        // postcondition: a customer is inserted in the hash table
        //-------------------------------------------------------------------
        void insert(Customer *c);
        {
                Check if the hash table is not full
                If the hash table is full
                        let the user know
                Otherwise,
                        Call hashFunc with the customerID;
        }
}
```

13. Class name: BST – Binary Search Tree to store the Movie in BST

```
// ---------------------------------- bst.h ----------------------------------------------------------
// Purpose: This file contains BST class function declaration
// ---------------------------------------------------------------------------------------------------------------
Class BST{
//-------------------------------------------------------------------------------------------------
// operation<<          : Overloads << to print BST
// precondition         : Movies have overloaded << operation
// postcondition        : Prints all movies inside BST
//-------------------------------------------------------------------------------------------------
friend ostream& operator<<(ostream &output, const Store& s)
{

}

Public:
//-------------------------------------------------------------------------------------------------
// BST()                : default constructor for BST class
// precondition         : NONE
// postcondition        : builds store object
//-------------------------------------------------------------------------------------------------
BST();
```

```
//-------------------------------------------------------------------------------------------------------
// ~BST()               : destructor for BST class
// precondition         : NONE
// postcondition        : BST is deleted
//-------------------------------------------------------------------------------------------------------
~BST();

//-------------------------------------------------------------------------------------------------------
// insert()             : insert Node with Movie data inside BST
// precondition         : movies have valid data
// postcondition        : Node with Movie Data is inserted to BST
//-------------------------------------------------------------------------------------------------------
bool insert(Movie *){
        if BST is empty
                add first Node with Movie Data
        else
                compare Nodes to find an appropriate location
                insert Node with movie Data
}


//-------------------------------------------------------------------------------------------------------
// find()               : find the location of the Node with correct Movie in BST
// precondition         : Movie has a valid data
// postcondition        : returns the location of the Node with the correct Movie
//-------------------------------------------------------------------------------------------------------
bool find(const Movie &, Movie *&){
        Traverse through BST with recursive
        If movie is found, return the location on 2nd Movie parameter
}

private:
        //-------------------------------------------------------------------------------------------------------
        // Node                 : Node to insert Movie to be used in BST
        //-------------------------------------------------------------------------------------------------------
        struct Node {
                Movie* movie;                   // pointer to data object
                Node* left;                     // left subtree pointer
                Node* right;                    // right subtree pointer
        };

        Movie* root;
```

```
};
```