CSS 343: Program 4 Design

Team: Sepehr Karimi, Biruk Haddis, Phuc Trieu Tran, Marcela Gomez

---

Overview

This system processes the business activities that are common to a movie rental store. The movie base class is inherited by the derived movie classes, namely drama, classics and comedy. In a similar fashion transaction base class has derived classes for the respective type of transactions in the store; borrow, return and history. From the “store” or “manager” class customers are created, movie inventory populated and transactions processing. Customer information is set and placed within a hash table with a randomly generated hash code that separates private data from the other accessible levels of the system.

Description of Main

Main will serve to create the inventory, customer list and transactions and then display each list accordingly from the incoming file information. It can also highlight some call functions within the methods are working such as, retrieve client, search movie, etc.

```
int main() {
        customerFile = “data4customers.txt”;
        transactionFile = “data4commands.txt”;
        movieFile = “data4movies.txt”;
        ourStore = new Store();
        ourStore.buildCustomerList(customerFile);
        ourStore.buildInventory(movieFile);
        ourStore.showStoreActivity();
        return 0;
}
```

Class Descriptions & Header Files

Class Client:

Description: This class processes information from the customer; including customer ID, first name and last name via a set data function that utilizes an ifstream to read-in the information from a text file. Using a hash function each customer will be assigned a

randomly generated key that stores the customer data into a hash table. Since a hash table is meant to create a layer to protect personal identifiable data from being exposed a hash table for a customer list makes the most sense. Two display functions will output either the customer data or show the customer's individual rental history. Our team plan to use a queue to store the customer history for each customer. As described in the directions the transactions will be stored in chronological order making a queue an ideal data structure as it prints FIFO ordering style.

```
class Client{
    // Constructor
    Client();
    Client(istream&);

    // Copy Constructor
    Client(const Client&);

    // Destructor
    virtual ~Client();

    // Set data from file input
    void setData(ifstream&);

    // Get Client ID
    int getCID() const;

    // Get name
    string getFirstName() const;
    string getLastName() const;

    // Transaction handling
    virtual void display();
    void recordTransaction(char);
    void displayClientHistory();

    // Hash function
    void hashTable();

    // Comparison operators
    virtual bool operator==(const Client&) const;
    virtual bool operator!=(const Client&) const;

private:
    int cid;
    string clientName;
    int size;
    HashTable *client[MINTABLESIZE];
    int hash(int);
    // Container to store client's transaction history
    vector<History> clientRentalHistory;
};
#endif /* Client_hpp */
```

Class Store:

Description: Store class creates a store object with its own list of customers, movies and list of its own transaction. Each list is inserted into a specific container as described in the private section of the class via an istream. A method that demonstrates what is in each of those lists. A client name can be searched. This class also uses the factory method that then uses the specific derived class to each base class, movie and transaction.

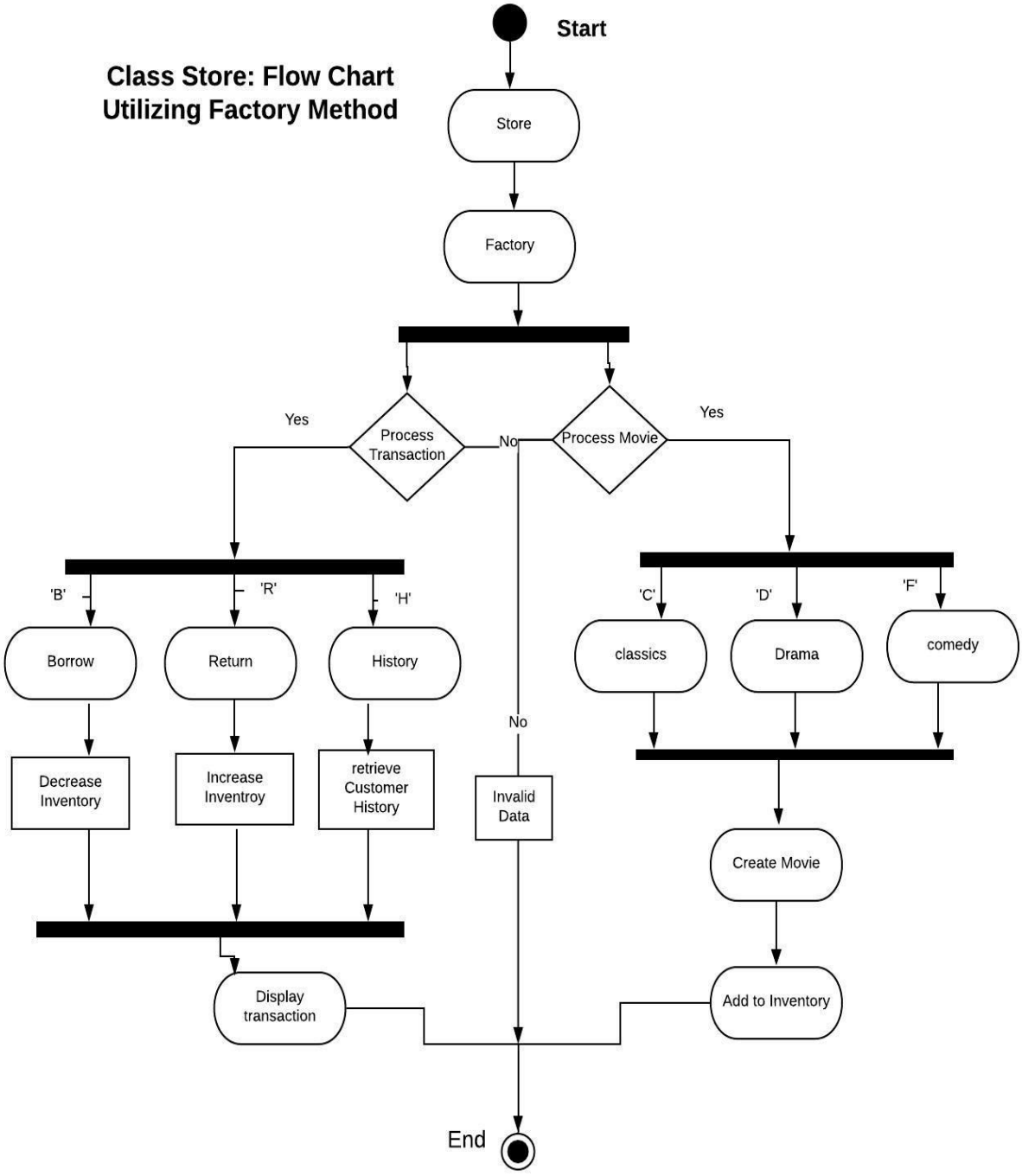
```
class Store{
public:
    // Constructor
    Store();

    // Copy Constructor
    Store(const Store&);

    // Destructor
    ~Store();

    // Build data containers
    void buildCustomerList(istream&);
    void buildInventory(istream&);
    void buildTransaction(istream&);

    // Display store's activities
    void showStoreActivity() const;
    void showCustomer() const;
    void showInventory() const;
    void searchClient() const;
    void getClient(const int) const;

private:
    // Constainer to hold store's data
    Client clientList[MAXCLIENTS];
    Bintree inventoryList[MAXINVENTORY];
    Factory factory;
};
#endif /* Store_h */
```

Class Factory:

Description: The factory method acts as a processing station in conjunction with virtual functions within classes. Much like a switch statement, when the factory method is called it instructs which movie class constructor or transaction method to call based on

the char in the incoming line from the text file. ‘D,’ ‘C,’ or ‘F’ would respectively call drama, classics or comedy classes to process information. ‘B,’ ‘R,’ and ‘H’ in turn indicate which of the three derived transaction classes to call for processing the transaction.

```
23  class movie;
24  class transaction;
25
26  using namespace std;
27
28  class factory()
29  {
30  public:
31
32      factory();
33      ~factory();
34
35      Inventory * processMovie(char, istream&);
36      Transaction * processTransaction(char, istream&);
37
38  private:
39
40      string mediaType = "DVD";
41      Transaction *factory[MAX_TRANSACTIONS];
42      Intentory * createMovieList
43
44  }
45
46  #endif //MOVIE_FACTORY_H
```


Class Store: Flow Chart
Utilizing Factory Method
Start
Store
Factory
Process Transaction
Yes
No
Process Movie
Yes
'B'
'R'
'H'
'C'
'D'
'F'
Borrow
Return
History
classics
Drama
comedy
Decrease Inventory
Increase Inventroy
retrieve Customer History
No
Invalid Data
Create Movie
Display transaction
Add to Inventory
End

Class Movie:

Description: Base class for all Classics, Drama, Comedy class of DVD type. This class inherited Inventory class, which provided attributes and methods to manage(view, update) number of available copies of a certain DVD/Videos.

```
9  class Movie : public Inventory {
10 public:
11     // Constructor
12     Movie();
13
14     // Copy Consstructor
15     Movie(const Movie&);
16
17     // Destructor
18     ~Movie();
19
20     // Set data for a movie item from file input
21     virtual void setData(istream& infile);
22
23     // Return information of a movie as string
24     virtual string getData() const;
25
26     // Return Director name
27     string getDirector();
28
29     // Return movie's title
30     string getTitle();
31
32     // Return movie's released year
33     int getYear();
34
35     // Comparison operators
36     virtual bool operator==(const Inventory&) const= 0;
37     virtual bool operator<(const Inventory&) const= 0;
38
39     // Create new movie item to the inventory
40     virtual Inventory* create() = 0;
41
```

```
41
42  protected:
43
44      // Movie's title
45      string title;
46
47      // Movie's director
48      string director;
49
50      // Released year
51      int year;
52      };
53  #endif /* Movie_h */
54
```

Class Comedy:

Description: A derived class of Movie to represent Comedy type DVD/Videos. It implements methods to read information of Comedy DVD and stock number from file input. Since it's derived class of Movie, it provides functions to store and update number of copies of a certain Comedy DVD of the shop.

```
 9  class Comedy : public Movie {
10  public:
11      // Constructor
12      Comedy();
13
14      // Copy Consstructor
15      Comedy(const Comedy&);
16
17      // Destructor
18      ~Comedy();
19
20      // Return information of a drama as string
21      virtual string getData() const;
22
23      // Set data for a comedy from file input
24      virtual void setData(istream& );
25
26      virtual bool operator==(const Inventory& ) const;
27      virtual bool operator<(const Inventory& ) const;
28
29      // Create new item to the inventory
30      virtual Inventory * create();
31
32  };
33
34  #endif /* Comedy_hpp */
```

Class Classics:

Description: A derived class of Movie to represent Classics type DVD/Videos. It implements methods to read information of Comedy DVD and stock number from file input. Beside common information with other kinds of DVD (title, director, year), it stores information about Actor and released date. Since it's derived class of Movie, it provides functions to store and update number of copies of a certain Classics Movie DVD of the shop.

```
 9  class Classics : public Movie {
10  public:
11      // Constructor
12      Classics();
13
14      // Copy Consstructor
15      Classics(const Classics&);
16
17      // Destructor
18      ~Classics();
19
20      // Return information of a classic movie as string
21      virtual string getData() const;
22
23      // Set data for a comedy from file input
24      virtual void setData(istream& );
25
26      virtual bool operator==(const Inventory& ) const;
27      virtual bool operator<(const Inventory& ) const;
28
29      // Create new item to the inventory
30      virtual Inventory * create();
31
32  private:
33      // Actor name
34      string actor;
35
36      // Released month
37      int month;
38  };
```

Class Drama:

Description: Similar to Comedy, this is a derived class of Movie to represent Drama type DVD/Videos. It implements methods to read information of a Drama DVD and stock number from file input. Since it's derived class of Movie, it provides functions to store and update number of copies of a certain Drama DVD/Videos of the shop.

```
 9  class Drama : public Movie {
10  public:
11      // Constructor
12      Drama();
13
14      // Copy Consstructor
15      Drama(const Drama&);
16
17      // Destructor
18      ~Drama();
19
20      // Return information of a drama as string
21      virtual string getData() const;
22
23      // Set data for a drama from file input
24      virtual void setData(istream& );
25
26      virtual bool operator==(const Inventory& ) const;
27      virtual bool operator<(const Inventory& ) const;
28
29      // Create new item to the inventory
30      virtual Inventory * create();
31
32  };
33  #endif /* Drama_h */
```

Class Inventory:

Description: Abstract class for Movie class. It declares functions to be overridden by its derived classes to read DVD information from file input, create new DVD/Video item, and comparison operators. It also provides functions and attributes to track and update stock number of each Inventory item.

```
7  class Inventory{
8  public:
9      // Constructor
10     Inventory();
11
12     // Copy Constructor
13     Inventory(const Inventory&);
14
15     // Destructor
16     virtual~Inventory();
17
18     // Set Inventory data from file input
19     virtual void setData(istream& ) = 0;
20
21     // Get Inventory's data
22     virtual string getData() const= 0;
23
24     // Increase stock number by 1
25     void increaseStock();
26
```

```
26
27     // Decrease stock number by 1
28     void decreaseStock();
29
30     // Get stock number
31     int getStockNumber();
32
33     // Comparison operators (by client ID)
34     virtual bool operator==(const Inventory&) const= 0;
35     virtual bool operator<(const Inventory&) const= 0;
36
37     // Create new Inventory item
38     virtual Inventory* create() = 0;
39
40 private:
41     // Store stock number
42     int stockNumber;
43 };
44 #endif /* Inventory_h */
```

Class Transaction:

Description: Is a base class for Borrow, Return, History classes. This class is the hub for the types of transactions that happen with the customer. It is a base class for Return, Borrow and History classes. It will also handle the error outputs for invalid action, video or customer ID and the different errors when borrow and return are invalid. This class is what gets printed when history is called.

```
5  class Customer;
6  class Transaction {
7
8  public:
9    Transaction(); // constructor
10   Transaction(const Transaction&); //copy constructor
11   virtual ~Transaction(); // destructor
12
13   virtual Transaction* create();
14
15
16   virtual bool setData(string media, Inventory* item, Customer* Customer);
17   virtual void display() const;// display
18
19   Inventory* getItem() const;
20   string getMovieType();
21   string getTransactionType();
22
23 protected:
24   string transactionType;
25   string movies;
26   Inventory* item1;
27 };
28 #endif
```

Class Return:

Description: Is a derived class of the Transaction class.This is denoted by **R** and Keeps track of the returned movies. Also kept track to add to the total number of movies in the inventory. If a movie was never borrowed and a return command is used, it will handle the error. All of the returns are kept track and will be displayed when History is called.

```
class Return: public Transaction{
public:
   Return(); //default constructor
   Return(const Return&);
   virtual ~Return(); //destructor
   virtual bool setData(string, Inventory*, Customer*);
   virtual void display() const;
   virtual Transaction* create();
};
```

Class Borrow:

Description: Is a derived class of the Transaction class.This is denoted as **B** and performs the action of borrow. When this is called, the stock is decreased by 1 to reduce the inventory. If the item is not available, it will ask other other alternative options for the customers. All of the borrowing transaction will be kept in track to make sure they are available when history is called.

```
class Borrow: public Transaction {
public:
  Borrow(); //constructor
  Borrow(const Borrow&); //copy constructor
  ~Borrow(); //destructor
  virtual bool setData(string, Inventory*, Customer*);
  virtual void display() const;
  virtual Transaction* create();
}
```

Class History:

Description: Is a derived class of the Transaction class.This is identified by **H** and prints the transaction history for the customer. It shows a list of DVD transactions of a customer in chronological order and specify whether the movie was borrowed or returned.

```
class History : public Transaction
{
public:
  History(); //default constructor
  History(const History& ); //copy constructor
  ~History(); //destructor
  virtual bool setData(string, Inventory*, Customer*);
  virtual Transaction* create();
};
```

Class BinTree:

Description: This is a node-based binary tree data structure which the object of Inventory class is passed as a node to this class. It has left and right subtree of a node. This class contains function to insert or remove a node into/from the binary tree. It also has functions to delete the tree by deleting all the nodes of the tree. showInorder function can display the content of bintree class in-order.

```
18  public:
19      BinTree();
20      BinTree(const BinTree&);
21      ~BinTree();
22
23      bool insert(Inventory* insertNode, const int recieved);
24      bool isEmpty() const;
25      void makeEmpty();
26      bool retrieve(const Inventory& movieData, Inventory*& targetData) const;
27      void retrieveHelper(Node* cPtr, const Inventory & movieData,
28                          Inventory*& targetData) const;
29      void showInOrder(Node * cNode) const;
30      Node * getRoot() const;
31  private:
32      struct Node
33      {
34          Inventory* movie;
35          Node* leftChild;
36          Node* rightChild;
37      };
38      Node *root;
39
40      void showInOrderHelper(Node* cNode) const;
41      void deleteTree(Node*& tree);
42  };
43  #endif
```

Pseudocode

1. **Methods invoked for building the collection of clients**

   void Store::buildCustomerList(fileName) {

```
        file = Get file from fileName;
        while (!file.end of file) {
                curLine = Get next line;
                Client cl = new Client(curLine.split);
                customerList.insert(cl);
        }
}
```

2. **Method for building Inventory. Read each line of the inventory file input to save to binary search tree, add stock number to existed DVD item if existed.**

```
        void Store::buildInventory(fileName) {
                fileinput = Get file from filename;
                while (!fileinput.end of file) {
                curLine = Get next line;
                switch(firstcharacter(curLine)):
                        case 'F':
                                Comedy f = inventorylist[hash('F')].retreive(Comedy(curline));
                                if(f is not null)
                                        Add stock number to retrieved comedy video
                                Else
                                        Create new Comedy Inventory
                        case 'C':
                                Classics c = inventorylist[hash('C')].retreive(Comedy(curline));
                                if(c is not null)
                                        Add stock number to retrieved classics video
                                Else
                                        Create new  Classic Inventory
                case D:
                                Drama d = inventorylist[hash('D')].retreive(Drama(curline));
                                if(d is not null)
                                        Add stock number to retrieved Drama video
                                Else
                                        Create new  Drama Inventory
                End switch
                }
        }
```

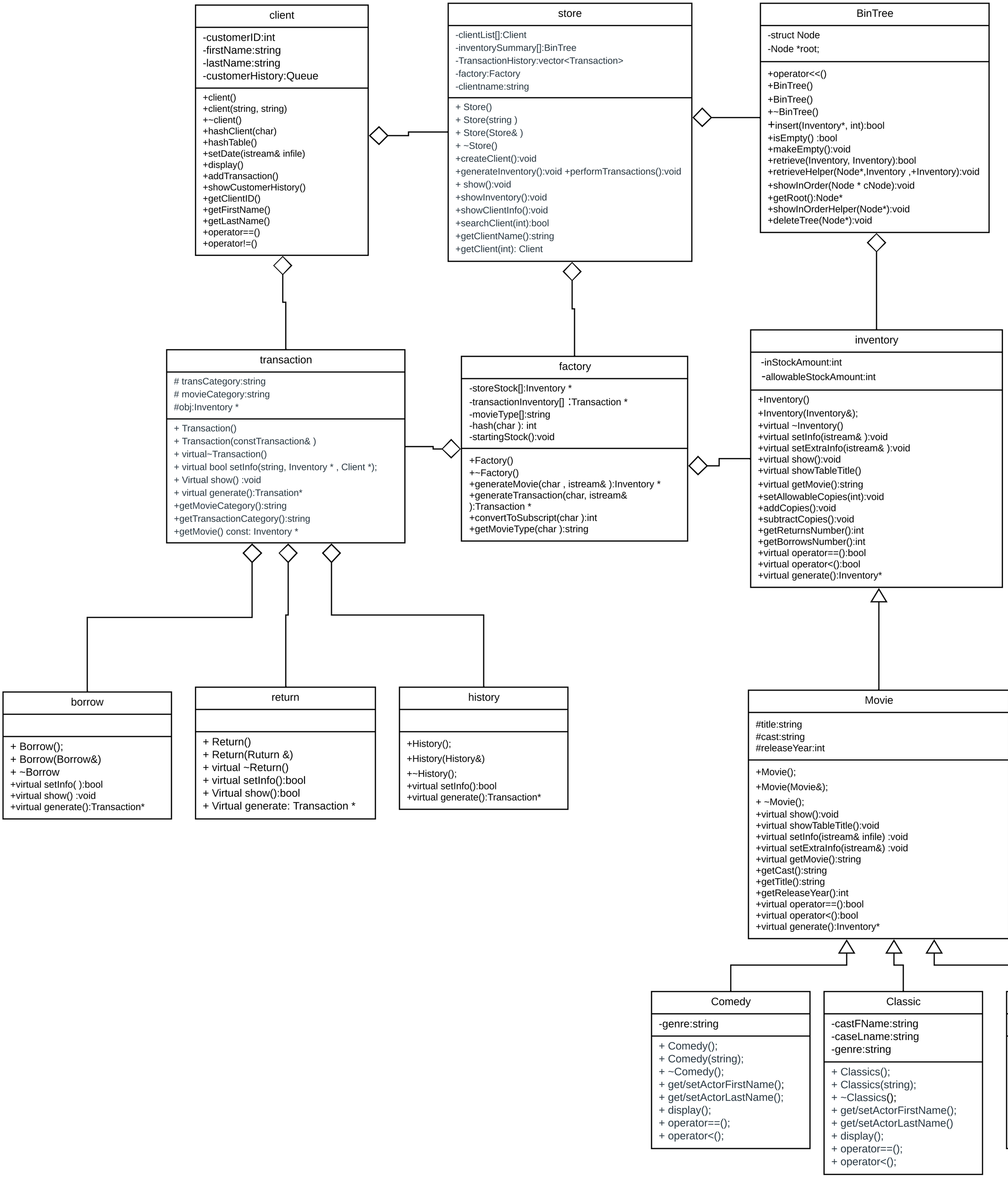
3. **Method to read commands/transactions to build transaction queue from file then execute all transaction in the queue to update Customer's rental data and store DVDs's stock number.**

```
void Store::processTransaction(fileName) {

        Queue  queue;
        for (;;) {
        tr = new Transaction;
        cmdLine = Get next line;
        successfulRead = tr->setData(infilecmd);
        if (not end of file) {
                if(successfulRead)   // good data
                        enqueue(tr) to transaction queue
                }
        }
        // Dequeue and execute transaction
        Transaction* tr
         while (queue is not empty) {
                queue.dequeue(tr);
                processTransaction
        }

}
```

4. **Method to process a Borrow Transaction**

```
bool Borrow::setData(string media, Inventory * item, Customer * aCustomer){

if(item!=null && item.getStockNumber()>1){

item->descreaseStock();

aCustomer->recordTransaction();

}

}
```

UML DESIGN


client
-customerID:int
-firstName:string
-lastName:string
-customerHistory:Queue
+client()
+client(string, string)
+~client()
+hashClient(char)
+hashTable()
+setDate(istream& infile)
+display()
+addTransaction()
+showCustomerHistory()
+getClientID()
+getFirstName()
+getLastName()
+operator==()
+operator!=()
store
-clientList[]:Client
-inventorySummary[]:BinTree
-TransactionHistory:vector<Transaction>
-factory:Factory
-clientname:string
+ Store()
+ Store(string)
+ Store(Store&)
+ ~Store()
+createClient():void
+generateInventory():void +performTransactions():void
+ show():void
+showInventory():void
+showClientInfo():void
+searchClient(int):bool
+getClientName():string
+getClient(int): Client
BinTree
-struct Node
-Node *root;
+operator<<()
+BinTree()
+BinTree()
+~BinTree()
+insert(Inventory*, int):bool
+isEmpty() :bool
+makeEmpty():void
+retrieve(Inventory, Inventory):bool
+retrieveHelper(Node*,Inventory ,+Inventory):void
+showInOrder(Node * cNode):void
+getRoot():Node*
+showInOrderHelper(Node*):void
+deleteTree(Node*):void
transaction
transCategory:string
movieCategory:string
#obj:Inventory *
+ Transaction()
+ Transaction(constTransaction&)
+ virtual~Transaction()
+ virtual bool setInfo(string, Inventory * , Client *);
+ Virtual show() :void
+ virtual generate():Transation*
+getMovieCategory():string
+getTransactionCategory():string
+getMovie() const: Inventory *
factory
-storeStock[]:Inventory *
-transactionInventory[] :Transaction *
-movieType[]:string
-hash(char): int
-startingStock():void
+Factory()
+~Factory()
+generateMovie(char , istream&):Inventory *
+generateTransaction(char, istream&
):Transaction *
+convertToSubscript(char):int
+getMovieType(char):string
inventory
-inStockAmount:int
-allowableStockAmount:int
+Inventory()
+Inventory(Inventory&);
+virtual ~Inventory()
+virtual setInfo(istream&):void
+virtual setExtraInfo(istream&):void
+virtual show():void
+virtual showTableTitle()
+virtual getMovie():string
+setAllowableCopies(int):void
+addCopies():void
+subtractCopies():void
+getReturnsNumber():int
+getBorrowsNumber():int
+virtual operator==():bool
+virtual operator<():bool
+virtual generate():Inventory*
borrow
+ Borrow();
+ Borrow(Borrow&)
+ ~Borrow
+virtual setInfo( ):bool
+virtual show() :void
+virtual generate():Transaction*
return
+ Return()
+ Return(Ruturn &)
+ virtual ~Return()
+ virtual setInfo():bool
+ Virtual show():bool
+ Virtual generate: Transaction *
history
+History();
+History(History&)
+~History();
+virtual setInfo():bool
+virtual generate():Transaction*
Movie
#title:string
#cast:string
#releaseYear:int
+Movie();
+Movie(Movie&);
+ ~Movie();
+virtual show():void
+virtual showTableTitle():void
+virtual setInfo(istream& infile) :void
+virtual setExtraInfo(istream&) :void
+virtual getMovie():string
+getCast():string
+getTitle():string
+getReleaseYear():int
+virtual operator==():bool
+virtual operator<():bool
+virtual generate():Inventory*
Comedy
-genre:string
+ Comedy();
+ Comedy(string);
+ ~Comedy();
+ get/setActorFirstName();
+ get/setActorLastName();
+ display();
+ operator==();
+ operator<();
Classic
-castFName:string
-caseLname:string
-genre:string
+ Classics();
+ Classics(string);
+ ~Classics();
+ get/setActorFirstName();
+ get/setActorLastName()
+ display();
+ operator==();
+ operator<();
Drama
-genre:string
+ Drama();
+ Drama(string);
+ ~Drama();
+ get/setActorFirstName();
+ get/setActorLastName()
+ display();
+ operator==();
+ operator<();