Overview

Our design is a UML Class diagram with the classes: Inventory, Customer, HashTable, Product interface, Media (interface), ProductDVD, DVD (interface), Drama, Classic, Comedy, CD,. To begin the breakdown of our UML, **Main** is our driver it will have an single instance of the Inventory class to represent the store. The Main function will have a way to read in the textfiles and identify the appropriate methods that need to be called to interact with the inventory. The second class we deal with is the **Inventory** class, this will work as the primary library connecting the rest of the components together. The Inventory has two instances of the **HashTable** class and some number of **Customers** illustrated by the bidirectional association in the UML. The remaining classes make up a factory method design pattern. The Inventory owns a number of Products represented by composition, all the products can be borrowed and returned by the inventory. The **Media** class declares an interface, which is common to all objects that can be produced by the creator and its subclasses. **DVD** is the first implementation of the Media interface and is denoted by realization between the two classes, itself is an interface from which the classes **Classic**, **Comedy**, and **Drama** inherited from (illustrated by realization on the UML). To further illustrate the abilities of the factory method we also included a **CD** class to represent a new category of rentable products. The **Product** class works as the creator class it declares the factory method that returns new product objects the Product class is unidirectional association to the media class. The final class **ProductDVD** and **ProductCD** are the concrete creator that override the base factory method to return a their respective type of product.

Class descriptions

Inventory

- Data Members
    - Private
        - media HashTable <Media>
            - This is a linked hashtable of type Media that uses open hashing, in the case of our inventory with only DVDs we will have 3 buckets for each category of DVD (Comedy, Drama, Classic).
        - customers HashTable <Customer>
            - This is a hashTable of customers that store customers using their unique customer IDs as values to define keys. This hashtable will use one of the closed hashing methods.
- Functions
    - Public
        - void Inventory()
            - Default constructor for Inventory class
        - void ~Inventory()
            - Deconstructor for Inventory class
        - void newMovies(ifstream&)

            - Adds new movies to the inventory from a provided data file ifstream
            - **Pseudo Code (Bellow)**

```
while (nextLine)
     ifstream >> type >> stock >> director >> title
     if (type == Classic)
          ifstream >> Major actor >> Release date
          media.insert(new Classic(stock,director,title,majorActor,
               release date);
     if (type == Drama)
          ifstream >> Year
          media.insert(new Drama(stock,director,title,year);
     else
          ifstream >> Year
          media.insert(new Comedy(stock,director,title,year);
```

        - newCustomers(ifstream&) : void
            - Adds new customers to the inventory from a provided data file ifstream
            - **Pseudo code (Bellow)**

```
while (nextLine)
     Ifstream >> CID >> fistName >> lastName
     customers.insert(new Customer(CID, firstName, lastName)
```

        - void printHistory(Customer)
            - outputs all the transactions of a customer in chronological order (latest to earliest) and specify whether the movie was borrowed or returned
        - void printInventory()
            - outputs the inventory of all the items in the store should be neatly formatted with one line per item/transaction. Output all Comedy movies, then all Dramas, then all Classics. Each category of movies should be ordered according to comedy movies sorted by Title, then Year it released, dramas are sorted by Director, then Title, classics are sorted by Release date, then Major actor.

HashTable

- Data Members
    - Private
        - hashChain *Node
            - This is the implementation of linked hashtable with an array base, the hashtable's will start as an array of Nodes with a number of buckets, as a collision occurs it will turn to a linked list where the root of the bucket's list is stored in the array. Open chain hashing.

- Functions
    - Public
        - void HashTable()
            - Default constructor for the HashTable
        - void ~HashTable()
            - Deconstructor for the HashTable
        - void insert (Obj)
            - To insert something into the hashtable
        - void delete (Obj)
            - To delete something from the hashtable
    - Private
        - int hashFunction(Obj)
            - The hash function to find the index at where the value should be stored

Customer

- Data Members
    - Private
        - String lastName
            - The last name of the current customer, it will always be a string.
        - String firstName
            - The first name of the current customer, it will always be a string.
        - int CID
            - The customer ID number, will always be an int. We chose to do an int over a String to make storing ID's in different data structures easier.
        - LinkedList transactionHistory
            - We decided to use a LinkedList to store the transaction history for each customer within the customer object itself. Because transactions will happen one after another and not simultaneously, we can simply create a new node and point the previous one to it.
- Functions
    - Public
        - void Customer()
            - Default constructor for the Customer object.
        - void Customer(int, String, String)
            - Constructor for new customer with (CID, firstName, lastName)
        - void ~Customer()
            - Default destructor for the Customer object. Will take care of clearing the LinkedList pointers, CID, and full name.
        - String GetFirstName()
            - Getter to get the firstName for the customer object because it is a private variable.

        - String GetLastName()
            - Getter to get the lastName for the customer object because it is a private variable.
        - int GetID()
            - Getter to get the customer ID for the customer object because it is a private variable.
        - void newTransaction(bool, Media)
            - Adds a new transaction to the transactionHistory LinkedList. Is passed the media being added to add to the LinkedList.
            - **Pseudo Code (Bellow)**

```
Node current = transactionHistory
while (current.next != null)
      current = current.next;
String newTransaction;
if(bool)
      newTransaction = "return " + Media.toString;
else
      newTransaction = "borrow " + Media.toString;
Node* newNode = new Node(newTransaction)
```

<<Interface>> Media

- Data Members
    - Private
        - stock int
            - The number of Media currently in the inventory (not borrowed)
- Functions
    - Public
        - void borrow(Customer, Media)
            - Borrows the Media to the passed customer. Will be reflected in the Customer's transaction history, effect the item's stock.
            - **Pseudocode (Bellow)**

```
Customer.newTransaction(false, Media);
Media.stock --;
```

        - void return(Customer, Media)
            - Returning the Media that was borrowed by the passed Customer, tracks it in the Customer's transaction history, replenishes the stock.
            - **Pseudocode (Bellow)**

```
Customer.newTransaction(true, Media);
Media.stock ++;
```

<<Interface>> DVD

- Data Members
    - Private
        - String title

        - Is the title of the DVD, will always be a String.
      - String director
        - Name of the director, will always be a String.
      - int year
        - The year that the movie was released.
- Functions
  - Public
    - void DVD()
      - Default constructor for DVD.
    - void ~DVD()
      - Destructor for DVD.
    - String getTitle()
      - Getter for the title of the DVD, which is a private variable of DVD.
    - int getStock()
      - Getter for the stock of the DVD, which is a private variable of DVD.
    - String getTitle()
      - Getter for the title of the DVD, which is a private variable of DVD.
    - String getDirector()
      - Getter for the director of the DVD, which is a private variable of DVD.
    - int getYear()
      - Getter for the year of the DVD, which is a private variable of DVD.

DVD::Drama

- Functions
  - Public
    - void Drama()
      - Default Drama DVD constructor.
    - void Drama(String, String, int)
      - Constructor for Drama class that initializes the variables: Title, director, year
    - void ~Drama()
      - Default destructor for Drama DVD.
    - ostream operator<<()
      - Prints Drama DVD.

DVD::Classic

- Data Members: classic has some additional data members unlike Drama or Comedy
  - Private
    - String majorActor()
      - The name of the major actor of the movie.
    - int month
      - The number of the month that the movie was released in.
- Functions

- public
  - void Classic ()
    - Default Classic DVD constructor.
  - void Classic (String, String, String, int, int)
    - Constructor for Classic class that initializes the variables: Title, director, majorActor, month, year.
  - void ~Classic()
    - Deconstructor for Classic DVD.
  - String getMajorActor()
    - Returns the major actor in this movie.
  - int getMonth()
    - Returns the month of release for this movie
  - ostream operator<< ()
    - Prints Classic DVD.

DVD::Comedy

- Functions
  - Public
    - void Comedy()
      - Default Comedy DVD constructor.
    - void Comedy(String, String, int)
      - Constructor for Comedy class that initializes the variables: Title, director, year
    - void ~Comedy()
      - Default destructor for Comedy DVD.
    - ostream operator<<()
      - Prints Comedy DVD.

<<Interface>> Product

- Function
  - Public
    - Media createMedia()
      - The Creator class declares the factory method that returns new product objects and returns the type of this method that matches the product interface.

ProductDVD

- Function
  - Public
    - Media createMedia()
      - Concrete Creators override the base factory method so it returns a product of type DVD