**Introduction**

The library management system is a program designed to efficiently handle the administrative processes and organization of library’s books and customers. Using object oriented principles, this system allows librarians to manage inventory, checkouts as well as management and creation of customers. put in place to allow librarians to manage the library’s inventory, check out and return items, and keep track of library members. Users have the ability to search for items by different ID or item title.

**Classes**

**LibraryMembers -** Users for the library system. Could be librarians or customers. Librarians have greater privileges than customers.

**Attributes:**

-libraryMemberID - Unique identifier for a library member. The library member could either be a librarian (who is an administrator) or a Customer.

-libraryMemberName – Name of the library member

-libraryMemberContact – Contact information for the library member

**Librarians (derived from LibraryMembers class)**

Within my library management system the librarians are the ones doing majority of the heavy lifting; meaning, that they are the ones performing the majority of the main functions in the library. They play more of an administrative role in the system and they are key players in the running of this system. This importance can be shown in the methods included in the library class.

**Attributes:**

-libID - Unique identifier for librarian

-libName – Librarian Name

-libContact – Librarian’s Contact Information

-list<Customers> customerList - List to hold customers

-list<libraryItems\*> itemList - List to hold library items

**Methods:**

-createAccount() - Has the ability to create a Customer Account

-viewCustomerList() - enables librarian to view the list of created customers and their details

- addLibraryItem() – Creates a library item (this could either be a book, journal, magazine or any form of digital media)

- displayAllItems() – This method displays all library items and their details.

-checkOut() – The librarian adds a chosen item to a chosen customer’s inventory.

-updateItemAvailability() – Method to update item availability after it is added to a customer's inventory

-searchList() – this function searches the list based on the item’s ID.

-returnItem() – This method removes an item from the customer’s inventory.

**CustomerAccount (Derived LibraryMember)**

**Attributes:**

-custID - unique identifier for customer

-custName – Name of customer

-custContact – Customer contact information

-list<libraryItems\*> inventory – This is a list for the customer’s inventory

**Methods:**

- displayCustomer() – Displays customer’s details

- list<libraryItems\*>& getInventory() – Getter for customer inventory

- addItemToInventory() – Adds library item to inventory

- displayCustomerInventory() – Displays items in customer’s inventory

**LibraryItem** (this has an associated relationship to Librarians and CustomerAccount.)

The library item is a parent class of the following child-classes: Book, Journal, Magazine and DigitalMedia. The library item class contains all relevant information for the child-classes.

**Attributes:**

itemID – unique identifier for the item

title – title of the item

author – author of the item

availStat – status of the item, whether it is available or not

**Book (Derived from LibraryItem):**

Attributes:

* bookISBN - Unique identifier for book

Methods:

* Constructor to initialize bookISBN
* Accessor and mutator for bookISBN

**Magazine (Derived from LibraryItem):**

Attributes:

* issueNo – Unique identifier for magazine

Methods:

* Constructor to initialize issueNo and call the parent constructor
* Accessor and mutator for issueNo

**Journal (Derived from LibraryItem):**

Attributes:

* journalID - Unique identifier for journal

Methods:

* Constructor to initialize ISSN and call the parent constructor
* Accessor and mutator for ISSN

**Digital Media (Derived from LibraryItem):**

Attributes:

* mediaFormat (e.g., DVD, Blu-ray, eBook)

Methods:

* Constructor to initialize Format and call the parent constructor
* Accessor and mutator for Format

**Relationships:**

**Librarians (derived from libraryMembers class):**

This class can inherit from LibraryMembers, which indicates an "is-a" relationship, i.e., a librarian is a type of library member.

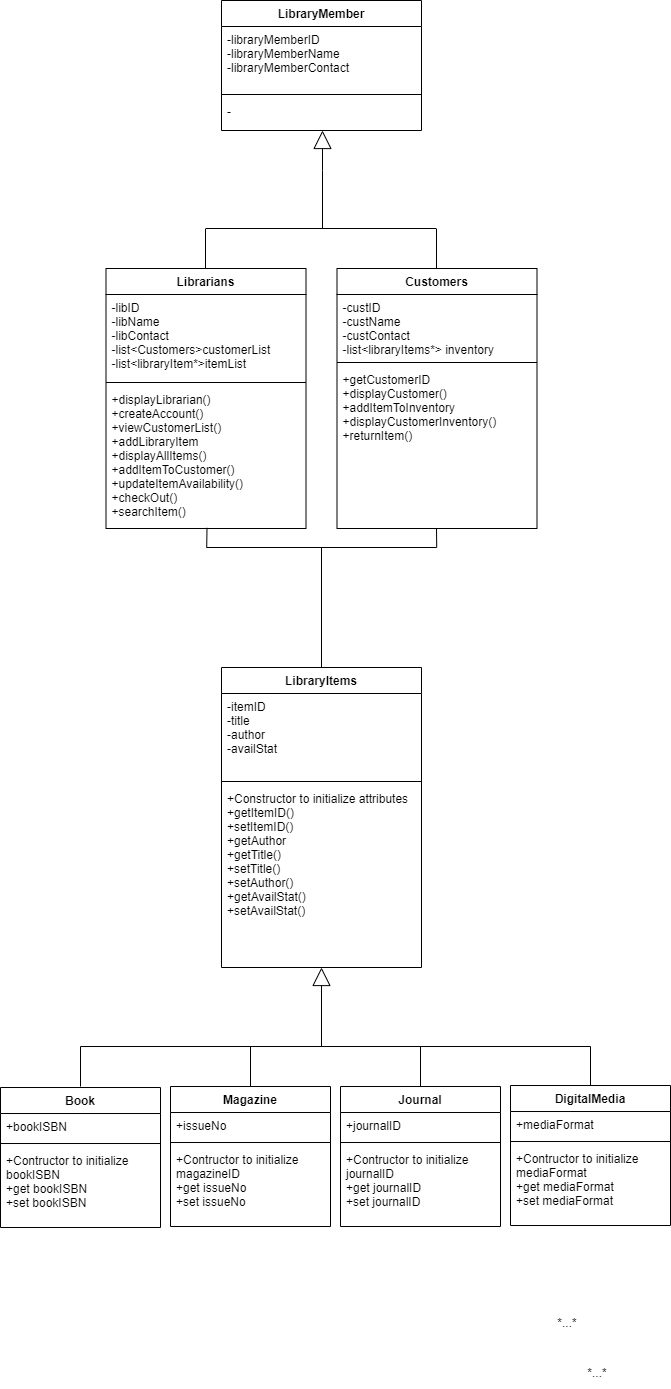
**Customers (Derived from libraryMembers):**

Similar to librarians, CustomerAccount can inherit from LibraryMember. Customers are a specialized form of library members.

**Book, Magazine, Journal, DigitalMedia (All derive from the LibraryItem Class):**

These derived classes represent specific types of library items, and they should inherit from the base LibraryItem class. This inheritance represents an "is-a" relationship.

**Class Diagram**



Documenting Code Design

**Edits**

* Initially, I envisioned my librarian class only being able to create customer accounts, add library items and approve checkouts. For my customer class, I envisioned the customer being able to choose from the list of available items, adding it to their cart and for it to be approved by the librarian. However, when it came to implementation, I realized that in order to do this there would need to be a noticeable separation between the librarian and customer accounts in the system. The separation I had in mind would have been the implantation of a register/login system. This register/login implementation would then involve the use of file handling and further complexities which I did not want to implement in the given timeframe. Seeing this, I changed the entire system around to giving the librarian more of an administrative role as to overseer role; where they are the ones who do the most of the functions in the library management system.
* Initially I planned on utilizing a doubly linked list without the use of the standard C++ library <lists>. This would involve me implementing a more manual approach in creating linked lists for my library item and customer classes. I then realized that a manual approach with the use of the doubly linked list proved to be quite complex. Seeing this, I switched to the singly linked. Things were going well in the beginning stages of implementing the list but midway through completion of this system I ran into some serious infinite looping problems. Seeing this: I would have made the switch to the standard C++ library since it emulates the doubly linked list (which is much more versatile) and it is simpler to implement (in my opinion).

**Additions:**

Data Structure – Linked Lists

* Compared to Stacks and Queues I feel like linked lists was the ideal pick since it shines when inserting or deleting elements anywhere in the list, making it ideal for situations where frequent addition or removal of items from the inventory or customer records occurs.
* In my opinion, I also think that compared to the other data structures, linked lists have a relatively simpler implementation, making them easier to manage and debug.
* In terms of traversing the list of items, I believe linked lists perform reasonably well. This feature might be beneficial when iterating through the inventory or customer lists. Which could be a great method to generate full-scale reports.

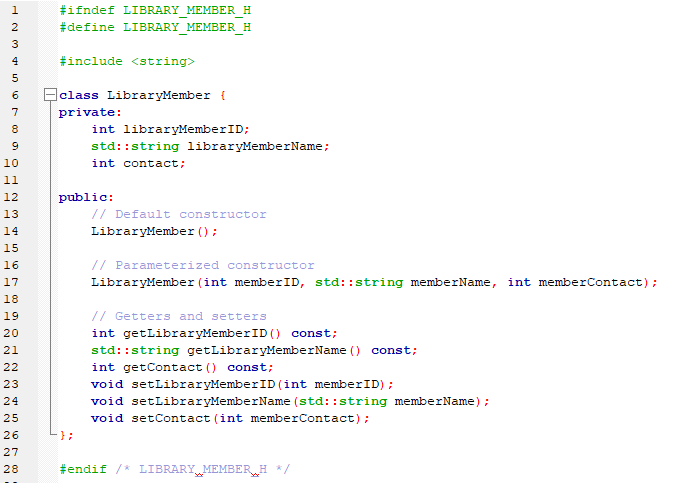
Object Oriented Programming Implementation

Inheritance – Define – Library Items and objects, Customer and LibraryMember

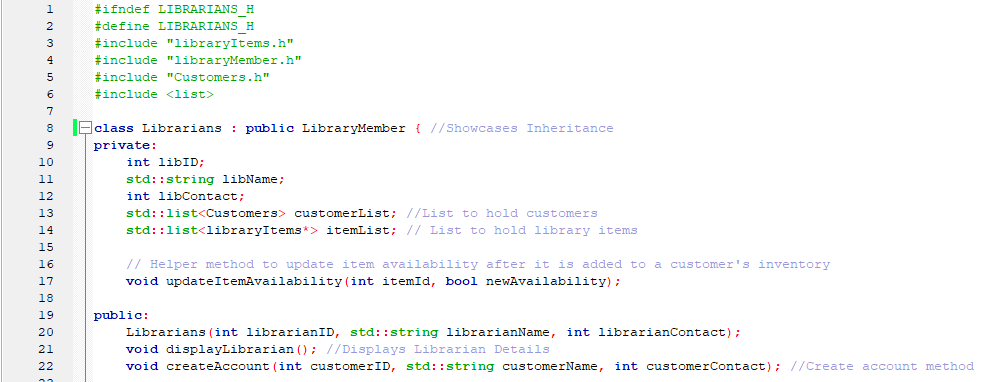
Polymorphism in objects of libraryitems

Implementation of Inheritance

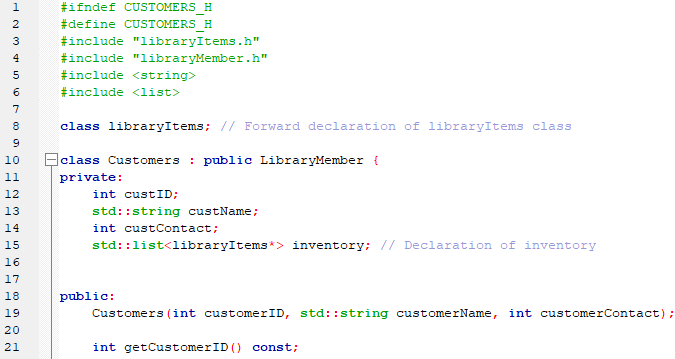
As discussed earlier, the libraryMember class is a parent class to the child classes: Librarians and Customers. I have included a screenshot of the header file below so you can get an idea of what the child-classes inherit from the parent class.



The below is a screenshot of my Librarians header file. If you take a look at **line 8**, you can clearly see that the Librarians class is being inherited from the LibraryMember class. Furthermore, you can see in **line 10** to **line 12**, that the Librarian class has also inherited classes from the LibraryMember class.

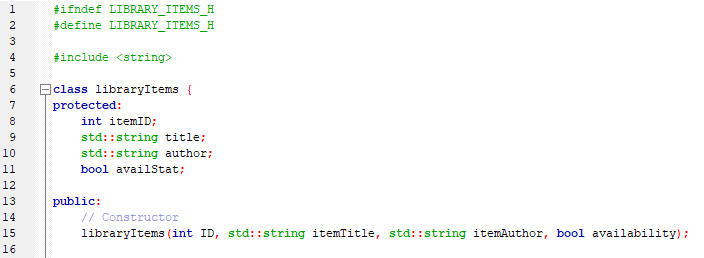


The same can be seen in the Customer.h file. I have included a screenshot of my Customer.h file which showcases inheritance at **line 8** as will as the inherited attributes from **line 12** to **line 14**:

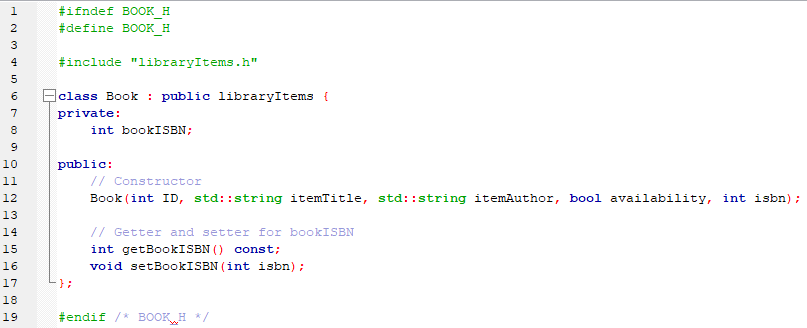


Implementation of Polymorphism

Below I have included a screenshot of my libraryItems.h file. As discussed earlier, the libraryItems class is a parent class to my Book, Jounal, Magazine and DigitalItems classes. This parent class includes the attributes which the above mentioned child-classes will inherit.

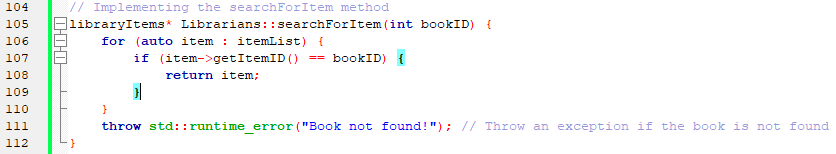


In the below screenshot of my Book.h file, we can clearly see polymorphism at play. If you look at **line 8** and **line 12**, you can see that we have successfully performed a function override of our libraryItems class by introducing and including the **bookISBN** attribute.



Search Algorithm Implementation

I have included a screenshot of my searching algorithm. I would have chosen the linear searching algorithm as it is straightforward and effective when dealing with small lists or unsorted data in this specific case. Unfortunately, it is not the most efficient algorithm for large datasets or scenarios where the items are sorted.



**Assumptions**

To be advised.