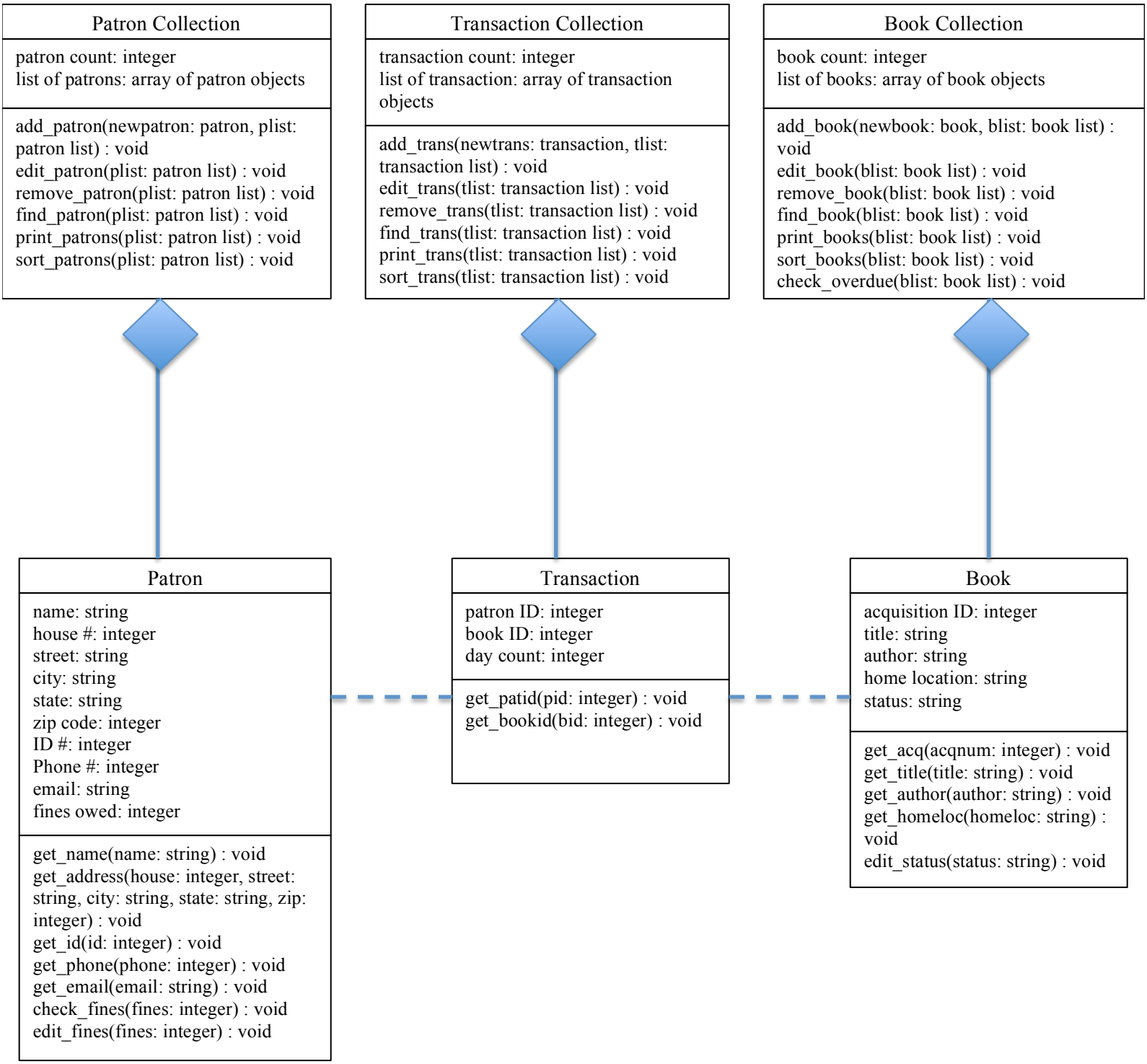
Cassiee Latshaw
CSCE 1040
July 14, 2014

Homework 4: Design for Smallsville Library System

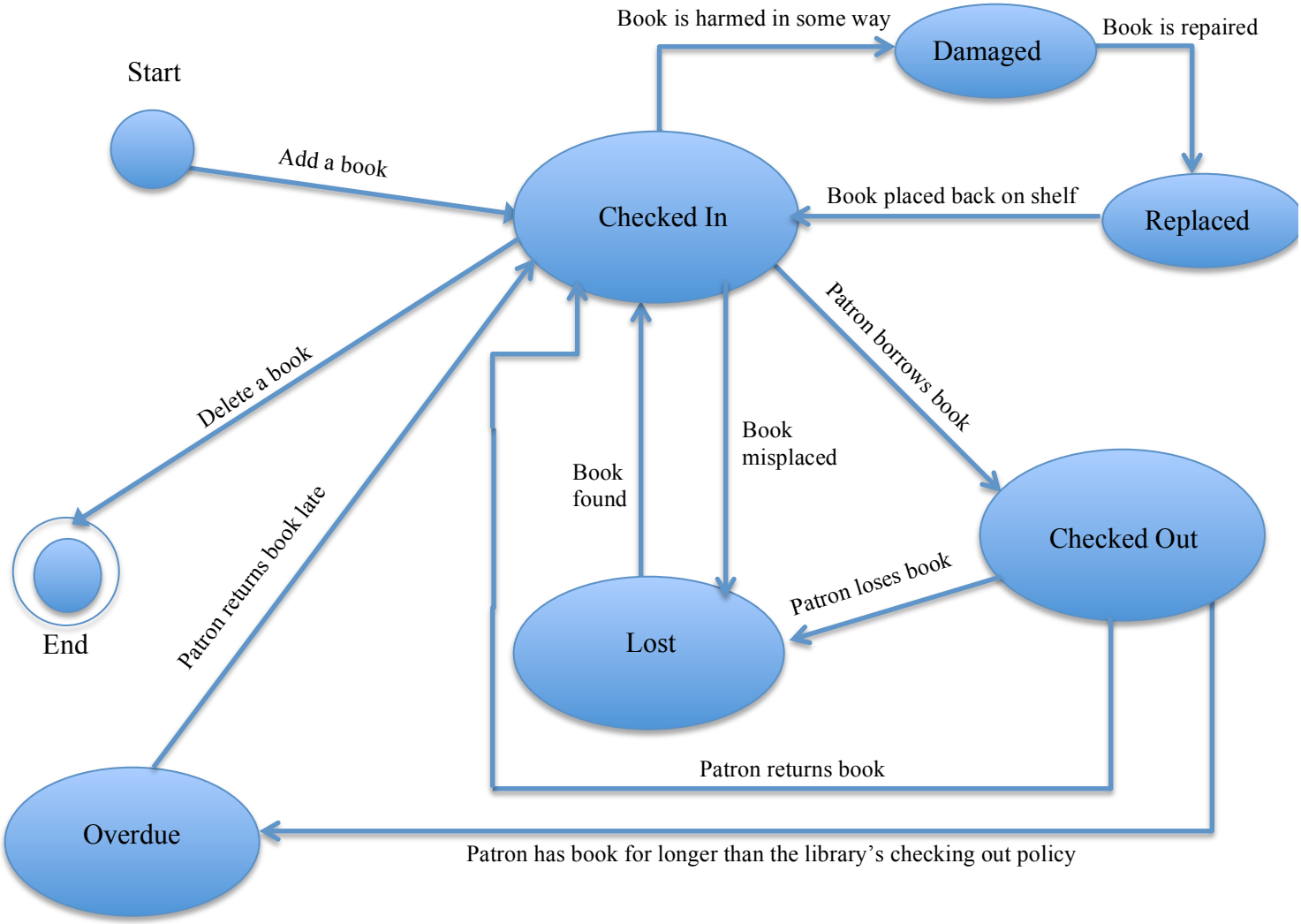
Design

Class Diagram for the Smallsville Library System



States Diagrams of Smallsville Library System

Book Status State Diagram:



Note: For the purposes of this assignment, I'll probably only use the checked in, checked out, overdue, and lost statuses for books. I'll maybe incorporate the damaged and replaced statuses on a later assignment (HW6 possibly).

Implementing the Diagrams for the Smallsville Library System

1. My general approach to this library system will try to incorporate a linked list with the transaction method. I used the transaction method in the previous assignment, and it seemed to make it easier when writing the program. I figured that if I used a linked list with this library system, adding and deleting patrons, books, and transactions would be easier since you could have the pointers point to something else (I didn't include the pointers in the class diagram on the first page because I'm indecisive on whether this is the route I want to go).

   Note: As I've started to think about how I want to do most of the functions, I'm not sure if I want to keep all of the function return types to be void (I've usually done void since when you change something by reference, it actually changes the contents of the variable), so I might change the return types and parameters of some of the functions. This was just a prototype (a rough draft, if you will) of how I'd want to do this program.

2. Whenever you start the program, it'll welcome you to the Smallsville library system and prompt the user to enter what they'd like to do with the program (I'll use a do…while loop so that the user can do as many things as they want without constantly restarting the program). Below are a list of options that I've brainstormed about what kind of functions the library system would need (but are also subject to change):

   a. Add a patron to the library system
   b. Edit an existing patron's information
   c. Remove an existing patron from the library system
   d. Add a book to the library system
   e. Edit an existing book's information
   f. Remove an existing book from the library system
   g. Print a list of all patrons (A-Z)
   h. Print a list of all books (A-Z)
   i. Print a list of all patrons with fines
   j. Print a list of all overdue books
   k. Print a patron's mailing address label
   l. Record a patron checking out a book
   m. Record a patron returning a book
   n. Record a book as lost
   o. Record that a book has been found
   p. Record that a patron paid a fine
   q. Store the library system to a file
   r. Load the library system from a file
   s. Quit

3. Once the user enters what they want to do with the program, it calls a specific function that carries out the user's desired choice.

a. Add a patron to the library system
    i. To add a patron, the program will call on the add_patron function of the patron collection, which will then prompt the user to enter all of the necessary information about that patron (using calls from the patron class).
        1. Once the user enters the patron's ID number, search through the current patron list to make sure that there isn't a patron that exists with the same ID.
    ii. The program then stores all of the information into the next patron object available, then increments the patron count.
b. Edit an existing patron's information
    i. To edit an existing patron's information, the user would need to determine which patron they'd like to edit
    ii. Prompt the user to enter the ID number of the patron they'd like to edit, and search through the patron collection until they find the patron with the correct ID number, and display all of the patron information onto the screen.
        1. If a patron with that ID number doesn't exist, print an error message
    iii. Prompt the user to enter which information they'd like to change (i.e. the address, email, etc.).
    iv. Then for whichever information they'd like to edit, the program calls the appropriate patron object function (i.e. get_address), and stores the new information into the previous variable (storing over it).
c. Remove an existing patron from the library system
    i. To remove an existing patron, the user needs to specify which patron they'd like to remove from the system.
    ii. Prompt the user to enter the ID number of the patron they'd like to delete, and search through the patron collection until they find the patron that has the matching ID number
        1. If there is not a patron with that ID, print an error message
    iii. Once you find the patron with the matching ID number, look at the transaction collection to see if there are any transactions that have that person's ID as the patron ID.
        1. If there are transactions with the corresponding ID, then that means that they have something checked out and you can't delete them until they return that book.
    iv. If there's at least one transaction with the patron's ID, print an error message saying that they have something checked out and can't be removed until they return their item.
    v. If there aren't any transactions with the patron's ID, prompt the user to make sure that they want to delete the patron.
        1. If they say yes, delete that patron object and somehow connect the two adjacent patron objects to each other, if

applicable (this is where I think that the linked list might be best for this); then decrement the patron counter.

d. Add a book to the library system
    i. To add a book, the program will call on the add_book function in the book collection, which will then prompt the user to enter all of the necessary information about that book
        1. Once the user enters the book's acquisition number, search through the current book list to make sure that there isn't already a book that has that acquisition number.
    ii. Then the program will store all of the information appropriately and increment the book counter.
    iii. Note: When creating a new book, you need to manually set the book's status to "Checked In".
e. Edit an existing book's information
    i. Prompt the user for the book's acquisition number and search through the book collection until it finds the book with the matching acquisition number.
        1. If no book is found with that acquisition number, print an error message.
    ii. Then the program will print all of that book's information and prompt the user to enter which part of the book's info they'd like to edit.
    iii. Then depending on what the user enters, the program will call on the appropriate function within the book class, and prompt the user to enter the new information and store it accordingly (storing it over the previous information).
f. Remove an existing book from the library system
    i. Prompt the user to enter the acquisition number of the book they'd like to delete, and search through the book collection until you find the book object with the same acquisition number.
        1. If there are no books with that acquisition number, print an error message
    ii. Once it finds the correct book, the program needs to check the status of the book.
        1. If it's checked in, it can be deleted.
        2. If it's checked out or overdue, it can't be deleted until it is returned, so display an error message.
        3. If it's lost, it can be deleted (I'm not sure about this one. I'll need to give it more thought).
    iii. If the book is checked in, the program will make sure that the user really wants to delete the book.
        1. If they say yes, then delete the information in that book object, then reconnect the two adjacent book objects, if able (once again, using a linked list); then decrement the book counter.
g. Print a list of all patrons (A-Z)

    i. To alphabetize the patron list, I'd most likely use a bubble sort (or another sorting algorithm) so that the list is in alphabetical order.
        1. Another option is to re-sort the list every time the user adds another patron (within the add_patron function) so that the list of patrons is already alphabetized for this function.
    ii. Then print out the list of patrons in alphabetical order.

h. Print a list of all books (A-Z)
    i. Just like the "print a list of all patrons" function, I was going to implement a sorting function (such as a bubble sort) so that the books were in alphabetical order (by title)
    ii. For future assignments, I could also possibly ask the user if they'd like to:
        1. Print a list of all books in the library system
        2. Print a list of all checked in books in the library system
        3. Print a list of all checked out books in the library system
        4. Print a list of all lost books in the library system
        5. Print a list of all books from a specific library (other than Smallsville)
        6. Print a list of all books written by a specific author
    iii. But for now, I'll just print a list of all book titles and authors in alphabetical order (and possibly their status to show the user where the book is)

i. Print a list of all patrons with fines
    i. To print a list of all patrons with fines, the program would go through all the patron collection, and call the "check_fines" function.
    ii. This function would simply see if a specific patron's fine balance was greater than 0.
        1. If it is greater than 0, then print the patron's name, ID number, and how much they owe in fines.
        2. If their fine is 0, then they don't have any fines.
    iii. Note: There should never be a patron's fine balance that is below 0 at any time.

j. Print a list of all overdue books
    i. To check to see if a book is overdue, the program would go through the book collection, and calls the "check_overdue" function.
    ii. This function would simply go through the book collection, and check the status of each book.
        1. If the status is "overdue", then print out the book's title, author, acquisition number, and how long the book has been checked out for.
        2. Then search through the transaction collection until you find the transaction that contains that specific book's acquisition number, and note the patron ID associated with that transaction

3. Then search through the patron collection until you find the patron with that ID, and print out that patron's name

iii. Note: I included a day count in each transaction, which is supposed to increment each day that the book has been checked out, but I'm not sure how to actually increment the day counter automatically
   1. I was thinking using some function in the time.h (or the C++ equivalent) header file to keep track of the time.
   2. Another option would be to manually ask the user to enter the date at the beginning of the program, and then calculate how many days the book has been checked out.

k. Print a patron's mailing address label
   i. To print a patron's mailing address label, prompt the user to enter the desired patron's ID number that they'd like to print the label for.
   ii. Then the program searches through the patron list until it finds the patron with the same ID number as what the user entered.
      1. If there is no patron with that ID number, print an error message.
   iii. Once the program finds the matching patron, print out the patron's house number, street, city, state, and zip code to the screen.

l. Record a patron checking out a book
   i. To record a patron checking out a book, prompt the user to enter the ID number of the patron they would be checking out that book.
      1. Search through the patron list to make sure that the patron actually exists. If they don't, print an error message.
   ii. Then prompt the user to enter the acquisition number of the book that patron would like to check out.
      1. Search through the book list to make sure that the book exists. If it doesn't, print an error message.
      2. Also, check the book's status. If it's "Checked In", then the patron can borrow it. If it's anything else (checked out, overdue, or lost), then they can't check the book out, so print out an error message.
   iii. Create a new transaction object within the transaction collection.
   iv. Then store both the patron's ID and book's acquisition number into that object.
   v. Increment the transaction counter.
   vi. And finally, search through the book list until you find the book with the matching acquisition number, and change that book's status to "Checked Out"

m. Record a patron returning a book
   i. To record a patron returning a book, prompt the user to enter the ID number of the patron that's returning the book and the book's acquisition number that they're returning.

1. Search through the transaction list until you find the transaction object that has both the same patron ID and book ID as what the user entered.
2. If such a transaction doesn't exist, print out an error message.

ii. Check the days counter on the transaction, and if it exceeds the number of days the library permits it to be borrowed (which in my case will be 21 days (at the moment), so 3 weeks), then print how much the patron owes in fines from the book being overdue.
1. Then search through the patron list until you find the patron with the matching ID number, and add the amount of fines they owe from returning the book to their current fine total.

iii. Then search through the book list until you find the book that matches the number the user entered, and change the status to "Checked In".

iv. Then delete the transaction object, and reconnect the two adjacent objects (hopefully with a linked list); then decrement the transaction counter.

n. Record a book as lost
i. To record a book as lost, the user needs to specify which book is now lost.
ii. Prompt the user to enter the acquisition number of the book they'd like to record as lost.
iii. Then search through the book list until you find the book with the matching acquisition number.
1. First, check to see what the current status of the book is.
a. If it is either "Overdue" or "Checked Out", then that means that the patron lost the book.
b. If the patron lost the book while borrowing it, then they need to be fined $15 for losing the book.
i. To do this, search through the transaction list until you find the transaction object that has the book's acquisition number, and then looks at the patron's ID associated with that transaction.
ii. Then search through the patron list until you find the patron with that matching ID number, and then add $15 to their current fine total.
2. Then change the status of that book to "Lost"
3. You can also say that the book is already recorded as lost by previously checking what the status of the book is.

o. Record that a book has been found
i. To record a book as found, the user needs to specify which book has been found.

ii. Prompt the user to enter the acquisition number of the book that has been found.
iii. Then search through the book list until you find the one that matches the acquisition number that the user entered.
    1. If the status is currently "Lost", change the status to be "Checked In" (which is the same thing as being found).
        a. Note: If the patron lost the book, but they found the book, they'll still owe the fine for losing it, and will still in a sense be returning it.
    2. If it's anything else (checked in, checked out, or overdue), print an error message saying that the book has already been located.

p. Record that a patron paid a fine
    i. To record a patron paying a fine, the user needs to specify which patron is actually paying for a fine.
    ii. Prompt the user to enter the patron's ID number that paid a fine.
    iii. Search through the patron list until you find the patron with the corresponding ID number.
        1. If there isn't a patron with that ID, print an error message
    iv. Then print out the patron's name, ID number, and their current total of fines that they owe.
    v. Prompt the user to enter the amount that the patron paid.
        1. If the patron paid less than their total amount, subtract what they paid from their total, and store the new total into their fines balance.
        2. If the patron paid more than their total amount, print an error message saying that they paid too much, and to re-enter the amount.
    vi. Once the patron paid the proper amount, print a confirmation about how much they paid, and what their new fines balance is.

q. Store the library system to a file
    i. I included this function in the program so that it's possible for the user to store the library system that they've already worked on to a file, so that they can work on it again at a later time.
    ii. This function prints the patron collection, book collection, and transaction collection (which in turn will print all of the patrons, books, and transaction) to a file so that they can load it the next time they start the program.

r. Load the library system from a file
    i. I also included this function in the program so that it's possible for the user to load a pre-existing library system (similar to our previous assignment) that already has patrons, books, and transactions, which enables them to add on to a system that they've already created instead of starting over every time they start the program.

ii. This function will primarily read in each class (stored from the "Store the library system to a file" function) and store it into the correct class.

s. Quit
   i. The program simply prints a message thanking them for using the program and to have a good day, and then quits the program.

Future Plans and Expansion for the Library System

Below are a few ideas about what I could do to make future assignments easier:

- Allow the user to search for books by either title or author (in a way, printing a list of books can sort of be treated like a search since it displays everything that's currently in the catalog)
- Allow for multiple libraries in a larger library system
  - In a sense, I've prepared for this by prompting the user to enter what the home location of the book is
- See what books a certain library has
- Allow for different kinds of items in the library
  - i.e. books, CDs, DVDs, audio books, reference books, etc.
  - Use inheritance
- Try to incorporate a self-check out system

Any Questions that I Might Have For the Program

For the most part, I don't really have any questions about this assignment (it's very similar to the grade book assignment). The main one that I have right now is what to include in each of the .h files. I know that I need to include main in the first .cpp file, and the function definitions in the other .cpp file. Then in the first .h file, I need to include the class prototypes for each of the entities I plan on using for this assignment. The thing I'm having difficulties understanding is what to include in the other .h file since we normally put the function prototypes in that .h file, but the function prototypes are included in the class prototypes (hopefully you can expand on what goes in the second .h file).

If I had any questions about the actual library system, the only thing I'm not entirely sure about is how to keep track of how long a book has been out (to check if it's overdue). I've brainstormed keeping a checked-out counter (of how many days the book has been checked out), or possibly even a return date (to compare with the current date, which I would ask the user or use a time function (if one exists)), but I'm not sure how to go about checking to see when a book is overdue, so I'd appreciate your advice on how you might you'd check to see if a book becomes overdue.