**End User Instructions**: The user has to pick an option between 1 to 8 corresponding to the option they want on the main menu. If you pick a number out of range, it’s fine since the program will prompt you for a number in range.

1. Searching Books: First it shows the user all of the books in the library. Then user is prompted input on the title of the book that they want to view. If they enter a valid title, it shows the book’s information, else it tells the user that the title that they entered does not exist in the library, and then returns them to the home screen. If there are no books in the library, then the program will tell you that the library is empty.
2. Issuing Books: Allows the user to issue a book to a student. First the user will enter the title of the book that they want to issue. The program will show the user the info on that book, and then show the list of students registered in the library that they can issue that book to. The students will be displayed in a numbered list, so just pick the number corresponding to the student that is checking out the book. Note that if there are no students or no books, then the program will tell the user that they can’t issue any books. Also if a book you typed in isn’t valid for some reason such as it doesn’t exist in the library, or it’s already checked out, then the program will tell you. When you successfully issue a book to a student, it’ll tell you.
3. Returning Books: Allows the user to return an issued book from a student. The program will display a list of records showing the book that was issued, and the student that it was issued to, in a numbered list. Input the number corresponding to the issued book and borrowing student in order to confirm that the book was returned. If no books have been issued yet, and this option was pressed, the program tells the user that no books have been issued yet, and just return them to the home screen.
4. Adding Books: Lets the user input in the title, author, isbn code, and then the number of pages of a certain book. After the user completes inputting information, the program will add that book into the library. This allows the user to issue that book to students, and it shows that new book, when the user wants to see all books in the library.
5. Deleting Books: The user is prompted to enter the title of the book that they want to delete. Then the user will delete that book from the library. If the title of the book that the user picks isn't in the library, then the program will tell the user that it doesn’t exist. Also if the book is already issued or has already been checked out by another person, then the program will tell the user that the book can’t be issued. This leads to them being redirected to the home page again, and so they have to pick the delete book option, and enter in another book title if they want to delete another book. Note that if there are no books in the library, the program will tell the user and prevent them from going forward in the book deletion process.
6. Adding students: Prompts input for creating the students first name, last name, and then their id number. After inputting this information, the program will register the student into the library, which allows the user to issue books to that student. If the user enters an existing student id, then the user will abort the operation and give an error to the user.
7. Deleting Students: Displays a numbered list of all of the students in the library, and then prompts input for which student they want to delete. The user just picks the number associated with the student that they want to delete, and then this deletes or removes the student from the library. Now the user won’t be able to issue books to that student.
8. Entering “8” for option 8, will just quit the program.

**Dev Instructions/Future Modifications:**

1. Storing Books: For storing books, we put them in a hash table with the book’s tile (tower cased) as the key, and then a data structure containing the book’s information as the value. While this is very useful for searching for books based on title, the problem with this is that we cannot store books with the same title. Books can sometimes have the same title, and this implementation of using the titles as keys won’t work. I think the solution or change could be using a database that queries for the title and returns all rows with that same title. This in turn could actually eliminate the need for a hash table data-structure.
2. Checking out books: A good program would probably implement time in it so that we could have functionality such as having deadlines of when the borrower would return their books. If they return it on time, then there’s no penalty, but if not there could be some penalty system such as 3 strikes, and they have to pay a fee or something like that.
3. User interface: The command line interface isn’t the most advanced, so possibly changing it to a graphical user interface that allows for point and click interaction would make it a lot easier to use. It’d make the screen a lot less cluttered since having many options on a command line screen makes it extremely cluttered. I think a GUI akin to a web page would probably be the most optimal and user-friendly for librarians to use.
4. Adding Book: An addition that could be made is obviously having duplicate titles, but also having a check to make sure that the user is either entering in a 10 or 13 digit ISBN code. Also making sure that code is unique or making code that isn't already in the database would be a major improvement to the BookLibrary class itself. Though maybe using an API from a major online library to make sure the book they’re adding in is real or at least checking if the ISBN is valid.
5. Editing books: Having the option to edit books would be a good addition to the program. Editing things such as the title, author, ISBN, and other attributes of the books. Also when editing books a good modification is to make sure the edits show up everywhere else the book object is present. Such as when you edit a book that’s been checked out, the new edits show up in the section that shows the checked out book.
6. Editing students: Having a feature to edit student information would be good. Though, the edits would just involve maybe adding a description or notes on the student, such as putting whether the student was good at returning their books on time or something like that. The program shouldn’t really be able to edit things such as names, age, and other personal information because that’s more involved with a registrar’s office that deals with student records.
7. Deleting students: Assuming students have to have an account to use the library, the deletion portion of the program seems good. Though if this library was a real school library, the addition and removal of students would be handled by another group of people rather than being managed by the librarian. We assume that all students can use the library, and so the library is only able to read from a database of all students and pick the student they are issuing a book to.
8. Deleting Books: The idea of deleting books should remain the same, but this time rather than dealing with a hash table, the user would be making commits and changes to a database of books instead. Again they’d likely input information for the title of the book or other information in order to delete said book.

**Thought Process and Ideas behind aspects of the program:**

* Computer Science Theory: The hardest part was thinking out and laying out the foundations about how the hash table was going to work and store its items. Learning the theory behind chaining and hashing in general was relatively easy, but the difficult part was figuring out how to implement it. Also it took a while, but actually understanding and then implementing merge sort was difficult, but I definitely found it valuable.
* Linked List: Making sure the nodes in the linked list also contained the value for the key was very important for aspects such as collision resolution. Having good foundational methods for the linked list such as checking if a node existed was very helpful in creating the hash table. Note that the linked list, nodes, and hash tables were created with template classes for flexibility. Though the only thing that’s limited is that the key for the hash table is intended to be a string since it only hashes strings.
* Hash Table: I made the hash table accept the title of the book as the key to simulate or make something similar to looking up or accessing a book or item based on its title. A feature that’s commonly seen on ecommerce sites. The hash table itself uses a lot of methods from the linked list. It was very helpful using methods from the linked list, and just relying on abstractions and code I already made. Things such as checking if a linked list was empty was my way of making sure a bucket was empty, or searching for a node based on its key was the way for searching if a key-value pair existed in a bucket. This idea of abstraction was emphasized as I was mainly relying on the fact that the methods I created for the linked list worked.
* Book Library: When creating this class I put myself in the view of another developer and how they would be interacting with the BookLibrary class if they used it in their own program. This is why I created functions that prompted input within the class so that others could easily add or remove elements from the library. Not only this, but I made the add functions as well in case there were situations where you’d want to test the library and not have to go through inputting stuff. Error handling is embedded in both for either case as well.
* Utilities file and text data files: The utilities file is a file for general purpose functions/algorithms that aren’t really related to the program, but are helpful in the implementation. Functions to lower case a string or sort a list aren’t closely related to a program about managing a book library, but they’re helpful when developing or implementing which is why I put them in a separate file for better organization. At the start of the program, it reads from data files that contain book and student information, and as a result it adds those books and student objects to the BookLibrary class instance. The reason I did this is because it makes it easier to demonstrate and show the capabilities of the program such as issuing and returning books if you already have books and students already in the library.
* Hash and sorting algorithms: I used the division or modular hash algorithm because not only was it simple, but it was compatible with the types of keys that my hash table was going to take. I already knew the hash table was going to take a string so the algorithm was a good fit. I used merge sort because I saw its time complexity was not only efficient, but also because I wanted to challenge myself and implement one of the more complicated sorting algorithms. I made it so that it sorts vectors and returns a vector because that helped me better understand the algorithm and theory since I was more familiar with lists in general.