Background of Tema International School

Tema International School, located in Ghana West Africa, is a secondary school that offers courses from grades 7 to 12. In Tema International School, students are prepared to write the International General Certificate of Secondary Education (IGCSE) exam from grades 7 to 10, and from grades 11 to 12 students are offered courses for the International Baccalaureate exam.

Established in 2003, Tema International School is still in its growth phase, and as such changes are continuously being made to the way activities are carried out in the school.

Problem Statement

With the many books in store, and the many people who visit the school library regularly, the librarian has the tedious job of managing the whereabouts of all the books that are borrowed, missing or currently available in the library. Thus, in Tema International school where the library is managed using a manual system in which everything is stored on paper, there is a high risk of information and books being lost, and also more resources have to be used in order to create storage space for the loads of paperwork that contains information regarding the details of books in the library.

For the librarian, the problem associated with managing the library manually is the large amount of paperwork that is involved. The librarian has to write the details (title, author, publisher, date of publication, index number) of all the books stored in the library in a notebook, and must also continually update it when new books are brought into the library. Also when books are being borrowed, the librarian must also record information like the details of the book being borrowed (title, author, publisher, index number), the date of borrowing, the date it will be returned, and details of the borrower (name, age, student number) into a separate book. Another problem the librarian encounters whilst managing the library occurs when the duration for borrowed books are extended; this is because the librarian then has to cancel the due-date previously recorded and rewrite the new due date for that particular book into the notebook that stores information for borrowed books, creating a mess in the notebooks for storing book details. Also, due to the large amount of books kept in the library, and the fact that details regarding these books are stored in notebooks, the librarian faces a lot of difficulty when details of a specific book have to be searched for.

However, it is not only the librarian that finds the present method of managing the library inconvenient, students also complain about the fact that this method of managing the library is not friendly towards them. Most students complain that due to the fact that information regarding borrowers of books are stored manually (by paper), the librarian usually makes mistakes when updating information regarding new borrowers of books, or forgets to update when a new student has borrowed a specific book; as a result, students complain that they get wrongly billed for misplacing books that they did not even borrow. Also, students complain that they have no way of checking what books are currently in the library or what books have been borrowed; this, they say affects their revision plans since they cannot effectively plan their studies due to the fact that they can never be confident that a book they would

need for their studies will be available in the library. Another problem students complain about is that as a result of the large number of books in the library, and the fact that details of library books are stored in notebooks, students do not have access to a list that can be easily searched to show them all the books of a similar category in the library, since there are occasions their assignments require that they produce a list of all the books of a certain category in the library. One example a student gave was that an assignment he once had, required that he produce a list of all the books in the library that were written by Anita Desai.

Another user that finds problems with the present system is the accountant; for the accountant, he complains about frequent disagreements with students and parents who argue that they have been wrongly billed for missing books in the library.

In a bid to look for help in solving this problem, the school management has looked at library management systems employed by other schools. In one school, every student is given a card that contains the student's personal information and the student's library history (information regarding the number of books borrowed, lost, and returned by the student), and every time the student borrows or returns a book, the student's library history is updated to match whatever transaction took place between the librarian and the student. Also, in this system, details of library books are stored in Excel worksheets which are regularly updated by the librarian.

However, it is not certain that this solution will be suitable for Tema International School since schools are resourced differently, and manage their libraries differently. Thus, to be able to design a solution that will suit the requirements of the users of Tema International School's library, the processes each user goes through when using the current library system have to be examined.

The processes the librarian has to go through whilst managing the library can be categorized into three parts. The first part deals with situations when the librarian has to re-stock the library with new books. In this process, the librarian receives the stock of books supplied from the school's stock department; after, he/she takes each book and writes down its corresponding details like the author's name, title of the book, the book's publisher, the date the book was stocked into the library, and the book's genre. After each book's details have been inputted into the notebook, it is given a specific book Id. Below is a sample data entry into the notebook for storing book details:

Book ID	Title	Author	Genre	Publisher	Date received (dd/mm/yy)
000_1	Village By The Sea	Desai, Anita	Fiction	Penguin Group	05/08/11
000_2	Morning Noon & Night	Sheldon, Sidney	Fiction	Scholastic	05/08/11
000_3	The Great Gatsby	Fitzgerald, F. Scott	Fiction	Scribner	05/08/11
000_4	So Long A Letter	Bâ, Mariama	Fiction	Pearson Education Limited	05/08/11
000_5	Fiela's Child	Matthee, Dalene	Fiction	The University of Chicago Press	05/08/11

The second part of the processes the librarian goes through comes during the stage when a book is being borrowed from the library. This stage begins with the borrower going to the librarian to inform her of the intended book to be borrowed; after, the librarian takes the book and inputs the details of the book being borrowed into the notebook that stores information about borrowed books. Then, the librarian stamps on the book the date it is to be returned, so as to serve as a reminder for the borrower as to when he/she should return the book. Finally, after the book is stamped, the librarian also writes the due-date of the book into the notebook; from there, the book can now be taken out of the library. The following is a sample data entry by the librarian in the notebook that contains information about borrowed books:

Name of Student	Title of book being borrowed	Book ID	Due date (dd/mm/yy)	Date returned (dd/mm/yy)	Overdue By (days)	Fine
Kwasi Gyasi	Othello	000_11	20/09/11			
Kwami Brown	Village By The Sea	000_9	25/09/11			
David Johnson	Things Fall Apart	000_15	01/10/11			
Rheba Parrisa	The Grand Design	000_8	05/10/11			
Temi Adiola	Physics For You	000_16	12/10/11			

The columns for "Date returned", "Overdue By", "Fine" are empty as these were instances when the books were borrowed and had not been returned yet.

The third part of the processes relates to the period when borrowed books are being returned. At the beginning of this stage, the borrower first hands the book to the librarian, and then the librarian enters the date the book was returned into the notebook, and stamps the return date of the book; finally, the librarian checks if the book was returned after its due-date, and if this is true, the librarian charges the borrower a fine, which has a flat rate of GH₵ 1. Also, at the end of each school term, the librarian looks through the notebook to find out the names of students who were fined for submitting overdue books, and these names are sent to the accounts office for the fines to be added to their semester bill. A sample data entry by the librarian for this process is shown below:

Name of Student	Title of book being borrowed	Book ID	Due date (dd/mm/yy)	Date returned (dd/mm/yy)	Overdue By (days)	Fine (GHc)
Mary Biney	The Lion King	000_22	05/10/11	09/10/11	4	2
Tony Owusu-mensah	Village By The Sea	000_9	05/10/11	05/10/11	0	0
Brian Junu	IB Chemistry	000_19	19/10/11	21/10/11	2	1
Eric Adebanjo	Ethan Frome	000_30	22/10/11	22/10/11	0	0
Joseph Sawyer	Knife Edge	000_35	22/10/11	30/10/11	8	4

For students, when they want to check if a book they are looking for has been borrowed or not, they first go to the librarian who checks through the notebook that contains the names of all the books available in the library; the librarian first looks through the notebook to find out whether the library has that book or not, and if the book is available, the librarian checks whether the book has been borrowed or not. If the book is borrowed, the librarian tells the student the date the book should be returned so the student can check later on that date to be able to borrow the book. The table below is shows a portion of the notebook used by the librarian during this process:

Book Title	Author	Genre	Publisher	Status (Available or Not)	Return date
Fiela's Chila	Matthee, Dalene	Fiction	The University of Chicago Press	Y	
Morning Noon & Night	Sheldon, Sidney	Fiction	Scholastic	Y	
The Great Gatsby	Fitzgerald, F. Scott	Fiction	Scribner	N	05/12/11
Village By The Sea	Desai, Anita	Fiction	Penguin Group	Y	

On the table, when an "N" is put in the column for "Status (Available or Not)" it means that the book has been borrowed and thus it must have a return date. However, if there is a "Y" in the column for "Status (Available or Not)" then the book is currently in the library and thus it cannot have a return date.

Another process that students go through comes when they want a list of the names of books of a particular category that the library has in store; in this process, the student goes to the librarian and mentions the category of books he/she is looking for, an example would be non-fictional books, then the librarian looks through his/her notebook for books of that category, and every time he/she finds a book of that category being looked for, he/she writes the name of that specific book on a piece of scrap paper. After the librarian has finished looking through the notebook, and finished writing down the names of the books on the piece of scrap paper, the librarian then hands out the list of books on the piece of scrap paper to the student. However, students complain that this process is tedious and time-consuming, and that the lists are often filled with errors. For this process, the librarian uses the same table shown in the paragraph just above.

Looking at the processes that the users of the library management system go through, it is important that the new system should be made up of a database that will contain all the information about books and students. This database will be searchable and can be queried, so that students and the librarian alike can look for specific books they want with ease. Also this database must be able to be updated easily so that the librarian will find it easy to update the statuses and other information concerning library books with ease. Finally, this new system should be able to perform calculations so that the risk of the librarian making mistakes when it comes to fining students will be removed.

Criteria for Success

For the program to be successful, it has to satisfy the needs of all those who will use it; this means that, it must both solve the problems that the users have with the current manual library management system, and must also improve parts of the old system. Thus, in this section of the document, I will outline the various features the new system must have in order for it to be regarded as a successful system.

After closely examining the processes the users go through with the current system, it has been concluded that the best type of system to solve the problems users have with the current system has to be a database. This is because, due to the large amount of data that the current library management system handles, it is important that the new system provides a way of easily managing and manipulating data; and since databases provide a way of easily working with large amounts of data, a database should be the best tool to be used to manage the library. With this database, the librarian will be able to store book details without the need for any additional storage space in the library since all this data can be stored on the computer; also the database will reduce the risk of data loss, since it is less likely for data to get lost or stolen from the computer as compared to if the data was stored in notebooks and kept in a storage room in the library. In addition, as the problem analysis has shown, the librarian's activities include tedious and time-consuming activities like searching data regarding book details; however, with a database these tasks will no longer be as tedious and time-consuming since data will be automatically searched by the database. Also, it can be seen that many of the disagreements between parents and the accountants are as a result of miscalculations by the librarian when fining students; thus this database will have the capability to carry out calculations for the librarian regarding issues like fines. This will eliminate the issue of arguments between parents and the accountants, since the database will be able to carry out these calculations more accurately. Also, since computer systems are considered to be more accurate at performing calculations than humans, parents will be less doubtful of the fines they are requested to pay once they get to know that these calculations were performed by computers.

With all this in mind, the new system will be centered on a database that can only be accessed by the librarian. The librarian will be the only person allowed to input and manage data to be stored in the database, this is due to the fact that information like the fines students have to pay can be tampered with by students if they are also given access to the new system. As the problem analysis has shown, there are times when students want use the library system for personal reasons, an example will be the times when students want a list of books of a certain category, however, even in these situations it will be the librarian who will search for these lists for the students, and hand a printout of the list to the students.

Based on the problem analysis made earlier, the program will be considered to be successful if it meets the following requirements:

The program should be able to store data

1. The program should be able to add details of new books to existing database of books, since the problem analysis shows that the librarian will need to add new book details during the period the library is being re-stocked.
2. The program should be able to record the following book details: book title, book publisher, book genre, and the date a particular book was received. This is important because, as the problem analysis shows, the librarian needs to enter those book details when the library is being re-stocked.
3. The program should be able to store student details in its database; students have to be stored in the database since they are the people who will be borrowing the books, and so information regarding them has to be stored.
4. The program should allow the librarian to input data like, "student Id", "book Id", and the date a borrowed book is due, during the stage a book is being borrowed from the library.
5. The program should allow the librarian to enter the date a book is being returned during the stage a borrowed book is being returned by a student.

The program should be able to allow data to be searched

6. The program should allow the librarian to search for specific books in the database because the librarian will need this feature during the stage when a book is being borrowed, since the book first has to be searched for, before the librarian can input data regarding the name of the borrowed book, and its due date.
7. The program should allow the librarian to search for names of students who have been fined; this is important since the librarian has to submit this list of names to the accounts office at the end of the school term.
8. The program should allow the librarian to search through the database to check whether a requested book exists in the library database or not, and should also be able to inform the librarian whether the searched book has been borrowed.
9. The program should allow the librarian to search for books of a particular category (author, publisher, or genre), and return a list of books within that category. This is important since the problem analysis shows that there are times students ask the librarian for a list of books that fall within a particular category.

The program should be able to allow data to be edited

10. The program should be able to change the "status" of a book being returned.

The program should be able to perform calculations

11. The program should be able to check if the book was returned after its due date; and if the book was returned when it was overdue, the program should be able to calculate the total fine the student has accumulated.

Another factor that will be used to judge the success of the new system will be its reliability; and for the system to be considered as reliable, the following requirements should be met:

12. The interfaces of the new system should be user-friendly so that the librarian can easily use the program and find it more comfortable to use than the old system
13. Output should also be easy to read since this will also make the librarian find the program easier to use than the old system
14. Searches should be fast so that the librarian and students alike will not find this program as time-consuming as the old system

However, this system has some limitations that will affect how it will be used; these limitations include:

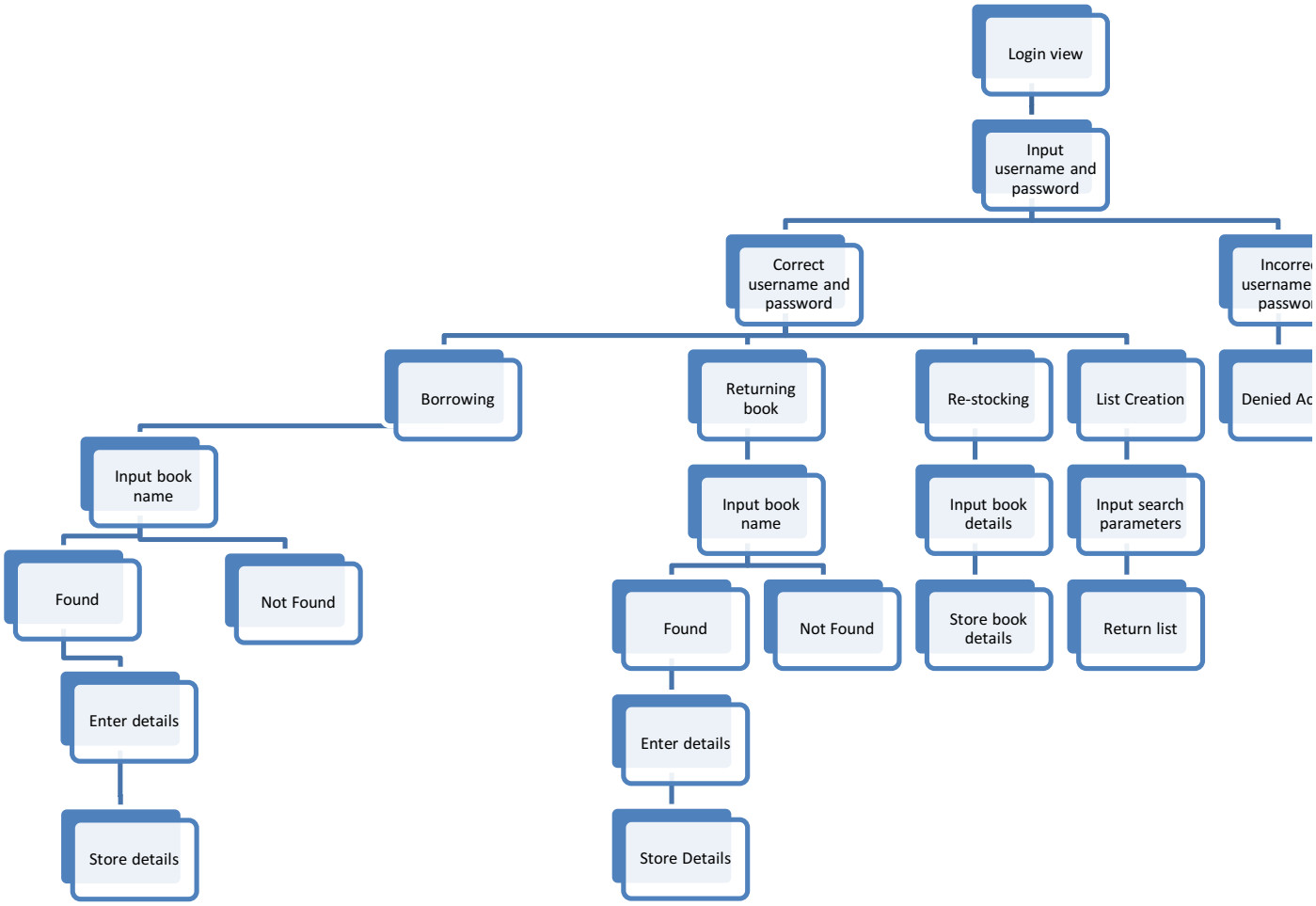
a) Due to the limitation of storage space, the program will only be possible to store the details of a 100 books and students in total.
b) Due to time constraints, the program will not use a graphical user interface, instead, the librarian will interact with the program through a command line interface.
c) The program will not be networked with other computers in the school; only the librarian's computer will have the program installed on it.
d) The program will require the use of a printer since there will be the need for the librarian to print out lists.
e) The program relies on the fact that Java is already installed on the computer it is being used on.
f) Students will only be allowed to borrow one book at a time, since giving them the chance to borrow more than one book at a time will make the program more complicated; this will make it difficult to complete the program within the deadline.

Prototype Solution

Based on the analysis and criteria for a successful program, an initial design for the program has been created; this design is centered on a database that will keep all the information about books and students in array lists. This is due to the fact that details of new books and new students will be continually added to the database of the program.

As it was stated earlier, only the librarian will have access to the program, and thus the program will always require the librarian to enter a username and a password before the program can be used; this is to prevent unauthorized users like students from getting access to the program's database and changing information like the fines that they have to pay.

Below are diagram showing the initial design of the intended program:



After careful consideration of the initial design for the program, the following prototype of the program was presented to the librarian:

Please enter your username: TIS Library

Please enter your password: *******

This represents the login view for the new program; as stated earlier, the program will not make use of a graphical user interface, and thus, the librarian will be interacting with the program through a command line interface. For the login view the first line that will appear will be the line asking the user to enter their username; since only one user is allowed access to the program, there will be only one correct username and password, thus the program will not continue unless the correct username is entered. After the correct username has been entered, the program will then ask the user to enter the correct password; just like the part where the correct username was asked for, the program will not continue unless the correct password has been entered.

```
Login successful

Are you borrowing, returning, re-stocking or creating a list?
```

After the correct username and password combination has been entered, the program will begin by asking the user to make a choice as to which of the four activities available (borrowing, returning a book, re-stocking or list creation) is the librarian about to undertake. Thus in this portal, the librarian is allowed to make only one of the four choices, any input other than the four choices will result in an error.

```
Welcome to the borrowing portal

Enter the name of the book to be borrowed: Fiela's Child

Book found!

Book Condition: Good

Name of Borrower: Kwaku Yeboah

Due Date: 23-08-11
```

This diagram above represents the page the librarian will be viewing when he/she inputs that a book is to be borrowed. The portal will begin by asking the librarian to enter the name of the book to be borrowed, and if the book is found in the library database, the program will ask the librarian to enter the name of the student borrowing the book. If the student who is borrowing the book is found to be on the library database, then that student can borrow the book, and the program then automatically calculates the due date of the book. However, if the student who is borrowing the book is not found on the library database, a profile of this student is created and added to the library database before the student can borrow the book.

Feedback from Librarian

After the prototype solution was shown to the librarian, and interview was conducted with the librarian to receive feedback concerning the program design and prototype sent to the librarian. Below is a transcript of the interview conducted with the librarian:

1. Librarian: Can the program be made to have a graphical user interface instead, since this will be more convenient?

Me: Sorry, but due to time constraints and lack of enough memory space, I cannot make the program run on a graphical user interface.

2. Librarian: Can students be given access to the program too, so that I do not need to carry out personal activities like creating lists, for them?

Me: This will mean that I will have to create separate accounts for students, and give them different levels of access; however, I do not think this is necessary since giving students access will require more memory.

3. Librarian: Using book Id's when searching through the database will be more efficient than using book names, since two books can have the same name but cannot have the same book Id's?

Me: Point well taken, I agree with you. This will be more efficient since the library can have two books of the same name with one borrowed and one available. If I allow you to search using names, it could lead to instances where database will show that one book has been borrowed by two different users at the same time.

4. Librarian: Instead of letting the program calculate the due date of a book for me, can you let me input the due date of books myself. This is because some students are more responsible than others, and I will not want irresponsible students to keep library books for a long time.

Me: Yes I can do that.

Revisions

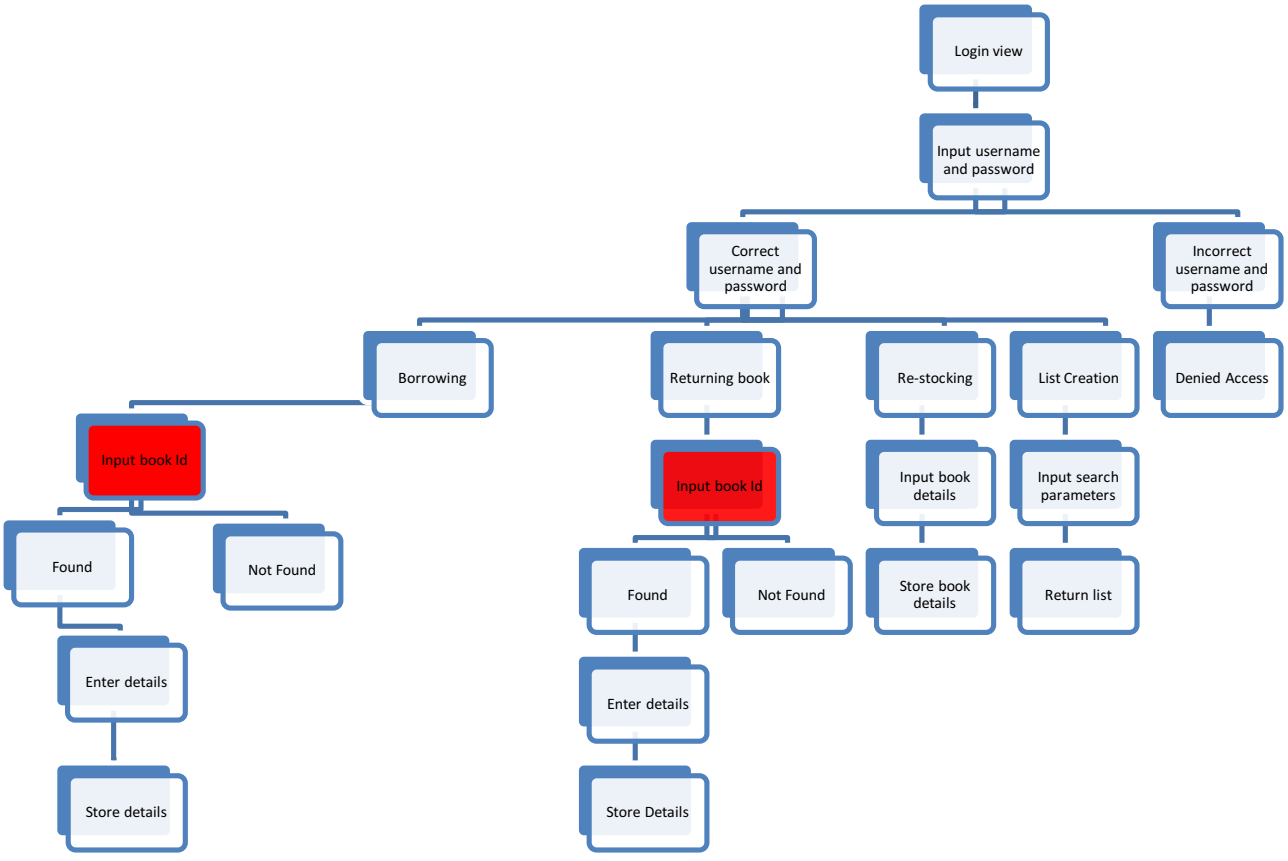
Based on the feedback received from the librarian during the interview, changes have been made to the goals and design of the solution.

Below, the changes made to the goals of the program have been outlined:

15. During the borrowing and returning stages, the program should allow the librarian to search for books in the library database using book Id's instead of book names

16. During the borrowing stage, the program should allow the librarian to enter the due dates of books to be borrowed.

Following the feedback from the librarian, these changes have been made to the program's design and its prototype solution:



Highlighted in red, the new design of the program shows that the program now allows the librarian to search for books by their Id's during the borrowing and returning stage, and not by book names again.

Part B- Data Structures

Array List

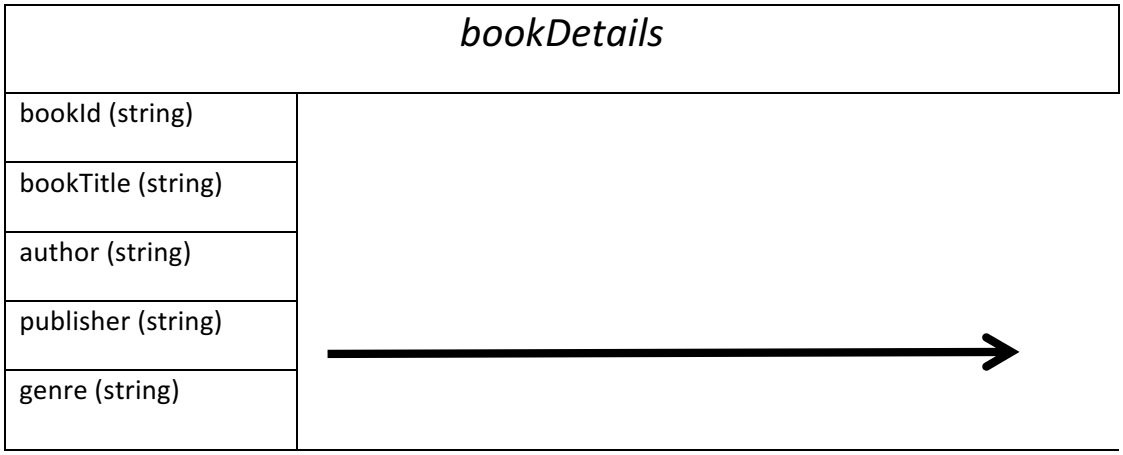
Book Details

An array list would be used to store the details of books. This will be used since many books are stored in the library, and thus it will be more advisable to store them in an array list than to store the details of `each book in a separate class. Also since new books are continually brought into the library, an array list will be suitable since it will allow data to be added to already existing lists.

The array list will allocate each entry to the details of a specific book. Thus, within each entry in the array list, the records for each book will be stored.

The fields that will store the records for each book are “bookId”, “bookTitle”, “author”, “publisher”, “genre”, “dateReceived”, and “status”.

The diagram below shows how the array list for book details will look like:



dateReceived (string)
status (String)

“bookId”, “bookTitle”, “author”, “genre”, “status” and “dateReceived” will be stored as string data types because calculations will not be done with the data stored in any of them.

Below is a representation of how the array list will look like when details of the books have been inputted:

bookDetails	
bookId: 000_001	bookId: 000_002
bookTitle: Village By The Sea	bookTitle: Morning Noon & Night
author: Anita Desai	author: Sidney Sheldon
publisher: Penguin Group	publisher: Scholastic
genre: Fiction	genre: Fiction
yearReceived: 2012	yearReceived: 2012
monthReceived: 01	monthReceived: 01
dayReceived: 01	dayReceived: 01
status: available	status: available

During the stage the library is being re-stocked, the librarian is allowed to enter data into all the fields available in the bookDetails array list, since these fields will be empty. However, after the details of the

new book have been entered, and the book has been added to the library database, the librarian will not be able to edit the data in the fields in that location. This process is represented below:

bookDetails	
bookId: 000_001 (locked)	bookId: 000_002 (locked)
bookTitle: Village By The Sea (locked)	bookTitle: Morning Noon & Night (locked)
author: Anita Desai (locked)	author: Sidney Sheldon(locked)
publisher: Penguin Group (locked)	publisher: Scholastic (locked)
genre: Fiction (locked)	genre: Fiction (locked)
yearReceived: 2012 (locked)	yearReceived: 2012 (locked)
monthReceived: 01 (locked)	monthReceived: 01 (locked)
dayReceived: 01 (locked)	dayReceived: 01 (locked)
status: available (free)	status: available (free)

As the diagram above shows, the after all the details for the books with Id's of "000_001", and "000_002" have been entered, the details of these books, excluding its status become locked, and thus the librarian cannot change the records of these books. The reason the why the status of a book remains free is because it automatically changes depending on whether it has been borrowed or not.; this also implies that librarian cannot change the status of a book, only the program can change the status of a book, and this depends on whether a book has been borrowed or not. Thus, when the books with the Id's of "000_001" and "000_002" become borrowed their statuses change to "unavailable". The diagram below shows this:

bookDetails		
bookId: 000_001 (locked)	bookId: 000_002 (locked)	bookId: 000_003
bookTitle: Village By The Sea (locked)	bookTitle: Morning Noon & Night (locked)	bookTitle: (free)
author: Anita Desai(locked)	author: Sidney Sheldon(locked)	author: (free)
publisher: Penguin Group (locked)	publisher: Scholastic (locked)	publisher: (free)
genre: Fiction (locked)	genre: Fiction (locked)	genre: (free)
yearReceived: 2012 (locked)	yearReceived: 2012(locked)	yearReceived: (free)
monthReceived: 01	monthReceived: 01	monthReceived:

		(free)
dayReceived: 01	dayReceived: 01	dayReceived: (free)
status: unavailable (free)	status: unavailable (free)	status: (free)

The diagram above also shows that when a new book is being added, the fields for the details of the book being added become free so that the librarian can enter data into these fields. The diagram above shows that a new book with an Id of "000_003" is being added to the library database, and thus the fields for this book have become free so that the librarian can enter data into them; but, as soon as the librarian has finished entering data into all the fields for the new book, and the new book is added to the library database, the fields for the added book, except its status, also become locked. The diagram below illustrates this:

bookDetails		
bookId: 000_001 (locked)	bookId: 000_002 (locked)	bookId: 000_003 (locked)
bookTitle: Village By The Sea (locked)	bookTitle: Morning Noon & Night (locked)	bookTitle: Things Fall Apart (locked)
author: Anita Desai (locked)	author: Sidney Sheldon(locked)	author: Chinua Achebe (locked)
publisher: Penguin Group (locked)	publisher: Scholastic (locked)	publisher: Scribner (locked)
genre: Fiction (locked)	genre: Fiction (locked)	genre: fiction (locked)
yearReceived: 2012 (locked)	yearReceived: 2012 (locked)	yearReceived: 2012 (locked)
monthReceived: 01	monthReceived: 01	monthReceived: 01 (locked)

dayReceived: 01	dayReceived: 01	dayReceived: 01 (locked)
status: available (free)	status: available (free)	status: available (free)

However, it must be noted that although the records of books added to the library database are locked, it does not imply that the data in these fields cannot be used by the program again; the fact that the data in the fields are locked just mean that the data in the fields cannot be edited by the librarian, however, this data can still be used by the librarian. Thus, the data for stored books in the library database will be used during the stage the librarian needs to search for specific books.

Student Details

The details about students will also be stored in an array list. This is due to the fact that the new student profiles will be continually added to the database, and thus an array list will be suitable since it allows new data locations to be added to it. Although the details of all of the students currently in the school could have been kept in an array with a fixed length, I preferred to use an array list as this will make the program more flexible since this will prevent the programmer from creating new arrays with new lengths to cater for new students admitted in the next academic year. Also a random access file could have been used since this would have allowed more space and thus more student profiles to be created; however, I preferred not to use this since using a random access file would have been more complex, and would hinder my ability to meet the deadline.

Each location in the student details array list will contain the following fields: “studentId”, “firstName”, “lastName”, “grade”, “bookBorrowed”, “yearDue”, “monthDue”, “dayDue”, “yearReturned”, “monthReturned”, “dayReturned”, and “totalFine”. Below is a representation of how the “studentDetails” array list will look like:

studentDetails	
studentId (string)	

firstName (string)
lastName (string)
grade (int)
bookBorrowed(string)
yearDue (int)
monthDue (int)
dayDue (int)
yearReturned(int)
monthReturned(int)
dayReturned(int)
totalFine(int)

"studentId" will store the specific Id of each student in the database, and the data stored in this field will not be altered by anyone. As a result of the data collected during the problem analysis stage, it was discovered that the Id of every student must begin with the sequence 'sd_', thus by making this field store only string values, the program will be able to ensure that whenever the librarian enters a student Id, it must begin with the sequence 'sd_'.

The "firstName", and, "lastName' fields will store the individual first and last names of each student in the library database; since a student's name cannot contain numbers, this field will store string values so that the program will be able to ensure that data entered into the fields for first name and last name do not contain numbers.

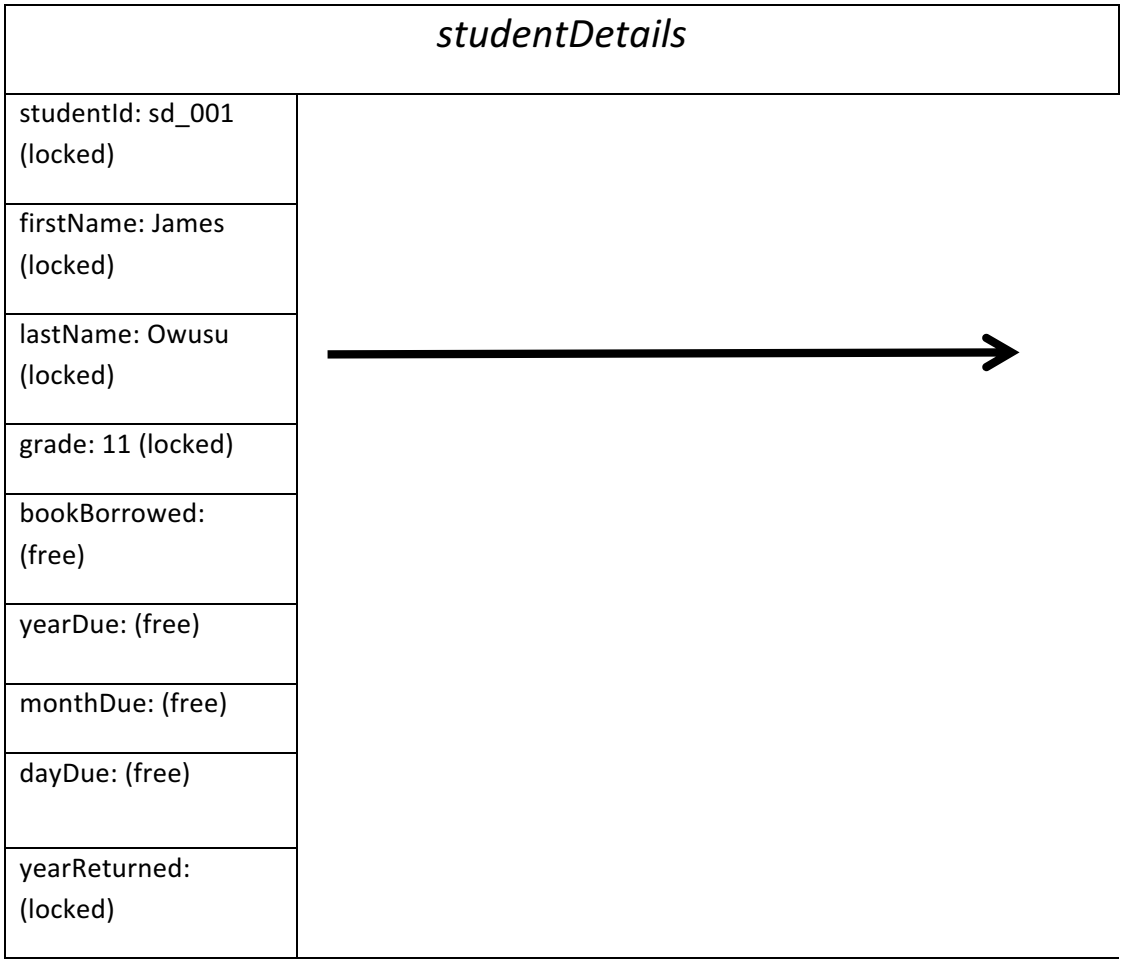
When a student borrows a book, the "bookBorrowed" field will store the Id of the book the specific student has borrowed; for this reason, this field will store string values since each book's Id must begin with the sequence '000_', and this data can only be stored as string values.

The 'grade" field will store the specific grade of each student in the database; data collected from the problem analysis showed that the school only runs from grades 7 to 12, and thus by allowing the data in this field to be stored as integer values, the program will be able to ensure that only the appropriate grades are entered by the librarian.

The fields for “yearDue”, “monthDue”, and “dayDue” will respectively store the year, month and day a borrowed book must be returned by a student. Therefore, these fields will only store integer values since the data stored in them will be later used to calculate whether a student should pay a fine or not.

The fields for “yearReturned”, “monthReturned”, and “dayReturned” will respectively store the year, month, and day a borrowed book was returned by a student; just like the fields for “yearDue”, “monthDue”, and “dayDue”, these fields will only store integer values since they will used in calculations when determining whether a student should pay a fine or not.

During the stage a book is being borrowed by a student, librarian can only input data into the “booKBorrowed”, “yearDue”, “monthDue” and “dayDue” fields since these are the only things that will be known to the librarian at that stage. At the stage a book is being borrowed, the librarian will have no idea as to the year, month, day, the student has returned the book; this will only be known to the librarian during the stage the borrowed book is being returned. Thus, during the stage a student is borrowing a book from the library, the fields of “yearReturned”, “monthReturned”, “dayReturned”, and “totalFine” become locked to the librarian whilst the fields for “bookBorrowed”, “yearDue”, “monthDue”, and “dayDue” become free to the librarian so that data can be inputted into these fields. The diagram below illustrates this:



monthReturned: (locked)
dayReturned: (locked)
totalFine: (locked)

As the diagram above shows, during the stage a book is being borrowed, the librarian will only be able to enter data into the "yearDue", "monthDue", "dayDue", and "bookBorrowed" fields whilst the rest of the fields in the "studentDetails" array list will become locked to the librarian.

One thing that must be noted is that the fields for "firstName", "lastName", "studentId", and "grade" will always be locked to the librarian once that specific student has been entered into the library database. Therefore, once a student profile has been added to the library database, the librarian cannot change the student's first name, last name, student Id, and grade; this is because these are pieces of information that do not regularly change- a student's first name, last name, and Id do not change until they have left the school whilst his/her grade changes only after a year has passed. Also, since the total fine a student has accumulated is automatically calculated by the program, the librarian will not be able to input data into the "totalFine" field; therefore this field will always be locked to the librarian. However, the other information in the student details array list regarding each student have the tendency to change regularly, thus none are this fields are made to be permanently locked. For example, the field for "bookBorrowed" can change on a weekly basis.

In the event that the profile of a student who intends to borrow a book does not exist in the library database, the librarian will the have option to add this student's profile to the "studentDetails" array list; however, during the stage a new student profile is being added to the "studentDetails" array list, only the fields for "studentId", "firstName", "lastName", "bookBorrowed", "yearDue", "monthDue", "grade", "dayDue", "yearDue" become free to the librarian- "yearReturned", "monthReturned", "dayReturned", and "totalFine" become locked to the librarian. The diagram below illustrates this:

studentDetails		
studentId: sd_001 (locked)	studentId: sd_002	
firstName: James (locked)	firstName: (free)	
lastName: Owusu (locked)	lastName: (free)	

grade: 11 (locked)	grade: (free)
bookBorrowed: (free)	bookBorrowed: (free)
yearDue: (free)	yearDue: (free)
monthDue: (free)	monthDue: (free)
dayDue: (free)	dayDue: (free)
yearReturned: (locked)	yearReturned: (locked)
monthReturned: (locked)	monthReturned: (locked)
dayReturned: (locked)	dayReturned: (locked)
totalFine: (locked)	totalFine: (locked)

The diagram above shows that a new student profile with the Id "sd_002" is being added to the "studentDetails" array, and thus the librarian can input data into the fields which have been marked as free whilst the fields marked as locked cannot be accessed by the librarian; this means that the librarian will not be enter data into data regarding the year a student returned the book, the day the student returned the book, the month the student returned the book, or the total fine a student has accumulated. And after the librarian has finished adding the new student profile to library database, the fields for that student's "firstName", "lastName", "studentId", and "grade" will always be locked to the librarian.

During the stage a student is returning a book, only the fields for "yearReturned", "monthReturned" and "dayReturned" become free to the librarian. The rest of the fields for that student remain locked to the librarian. Thus, in the event that the student with the Id "sd_001" is returning a book he borrowed from the library, the librarian will only be able to input data into the fields for "yearReturned", "monthReturned", and "dayReturned" for that student. The diagrams below illustrate how the array list changes before and after a book is returned:

studentDetails

studentId: sd_001 (locked)	studentId: sd_002	
firstName: James (locked)	firstName: (free)	
lastName: Owusu (locked)	lastName: (free)	
grade: 11 (locked)	grade: (free)	
bookBorrowed: 000_001	bookBorrowed: (free)	
yearDue: 2012	yearDue: (free)	
monthDue: 09	monthDue: (free)	
dayDue: 05	dayDue: (free)	
yearReturned: (locked)	yearReturned: (locked)	
monthReturned: (locked)	monthReturned: (locked)	
dayReturned: (locked)	dayReturned: (locked)	
totalFine: (locked)	totalFine: (locked)	

The diagram above shows that the student with Id "sd_001" has borrowed a book with Id "000_001", and that the student is supposed to return the book on the 5th of September 2012. As the book is not being returned yet, the "yearReturned", "monthReturned" and "dayReturned" fields are locked to the librarian. However, during the stage the book is being returned then the data in array list for that student becomes:

studentDetails		
studentId: sd_001 (locked)	studentId: sd_002	

firstName: James (locked)	firstName: (free)	
lastName: Owusu (locked)	lastName: (free)	
grade: 11 (locked)	grade: (free)	
bookBorrowed: null (locked)	bookBorrowed: (free)	
yearDue: 0 (locked)	yearDue: (free)	
monthDue: 0 (locked)	monthDue: (free)	
dayDue: 0 (locked)	dayDue: (free)	
yearReturned: (free)	yearReturned: (locked)	
monthReturned: (free)	monthReturned: (locked)	
dayReturned: (free)	dayReturned: (locked)	
totalFine: (locked)	totalFine: (locked)	

As the diagram above shows, during the stage a book is being returned the librarian can then input data into the"yearReturned", "dayReturned", and "monthReturned" fields as they become free, whilst the rest of the fields remain locked to the librarian. For the "bookBorrowed", "yearDue", "dayDue", and "monthDue" fields the program automatically sets them to 'null', and '0' respectively, to show that the student does not have a borrowed book in his/her possession anymore.

Sequential Files

The details of each student and book in library will have to be stored for permanent use, the profiles of students and books will be stored in separate sequential files. This implies that all the data about the books and students who use the library, which are stored in array lists will eventually be saved in sequential files. Although a random access file could have also been used, the creation of a random access file will be more complex and thus will prevent me from finishing the program within the

deadline; for this reason, I decided that the profiles of books and students will be stored in separate sequential files.

This implies that when the program is running, all the student profiles and book profiles created will first be stored in the array list, and this will later be saved in their respective sequential files. Therefore, the all the data in the "studentDetails" array list will be stored in an sequential file called "newStudentDetails" whilst all the data in the "bookDetails" array list will be stored in a sequential file called "newBookDetails".

This also implies, that for the program to be able to make use of stored book profiles and student profiles, all the data in each sequential file will have to be loaded into the program before these data can be used by the program. This means that, as soon as the program begins, the data in each sequential file will be copied into their respective array list so that new profiles can be added to the library database, and updates can also be made to already existing profiles in the library database.

Below are representations of how data will be transferred from the array lists into their respective sequential files:

bookDetails	
bookId: 000_001	bookId: 000_002
bookTitle: Village By The Sea	bookTitle: Morning Noon & Night
author: Anita Desai	author: Sidney Sheldon
publisher: Penguin Group	publisher: Scholastic
genre: Fiction	genre: Fiction
yearReceived: 2012	yearReceived: 2012
monthReceived: 01	monthReceived: 01
dayReceived: 01	dayReceived: 01
status: available	status: available

000_001	Village By The Sea	Anita Desai	Penguin Group	Fiction	2012	01	01	available
000_002	Morning, Noon & Night	Sidney Sheldon	Scholastic	Fiction	2012	01	01	available

The diagram above shows that each individual profile in the "bookDetails" array list will be stored in a line in the sequential file for books ("newBookdetails); the diagram also shows that a tab character will be used to separate the sections of each profile from each other. Therefore as the diagram shows, when the profile of the book with the Id "000_001" (highlighted in red) is being copied into the sequential file, its details will be stored on the first line of the "newBookDetails" sequential file. Then the next book profile (highlighted in yellow) will be stored on the next line in the "newBookDetails" sequential file. This also implies that when the "newBookDetails" sequential file is being loaded into the program, each line in the "newBookDetails" will be stored in one data location in the "bookDetails" array list of the program.

Also when a new book profile is to be added to the library database, the details of the new profile are first added to the "bookDetails" array list. Thus if a new book with an Id of "000_003" is to be added to the library database, the "bookDetails" array list will look like the diagram below:

bookDetails			
bookId: 000_001	bookId: 000_002	bookId: 000_003	
bookTitle: Village By The Sea	bookTitle: Morning Noon & Night	bookTitle: Things Fall Apart	
author: Anita Desai	author: Sidney Sheldon	Author: Chinua Achebe	
publisher: Penguin Group	publisher: Scholastic	publisher: Scribner	
genre: Fiction	genre: Fiction	genre: Fiction	
yearReceived: 2012	yearReceived: 2012	yearReceived: 2012	
monthReceived: 01	monthReceived: 01	monthReceived: 01	
dayReceived: 01	dayReceived: 01	dayReceived: 01	

status: available	status: available	status: available

Now if this new profile is to be added to the “newBookDetails” sequential file, the details of each profile are stored in a string variable, and in this string variable a tab character will be used to separate each section of a book’s profile; after, the data in this string variable will be copied into the “newBookDetails” sequential file, and that is how data will be saved into the “newBookDetails” array list. The illustrations below show this:

bookDetails		
bookId: 000_001	bookId: 000_002	bookId: 000_003
bookTitle: Village By The Sea	bookTitle: Morning Noon & Night	bookTitle: Things Fall Apart
author: Anita Desai	author: Sidney Sheldon	Author: Chinua Achebe
publisher: Penguin Group	publisher: Scholastic	publisher: Scribner
genre: Fiction	genre: Fiction	genre: Fiction
yearReceived: 2012	yearReceived: 2012	yearReceived: 2012
monthReceived: 01	monthReceived: 01	monthReceived: 01
dayReceived: 01	dayReceived: 01	dayReceived: 01
status: available	status: available	status: available

Therefore if the details of the book with Id “000_003” are being added to the “newBookDetails” sequential file, the string variable (bookLine) that will copy these details will look like:

bookLine: 000_003 Things Fall Apart Chinua Achebe Scribner Fiction 2012 01 01 available

After, the data in "bookLine" will be copied into the "newBookDetails" sequential file in order to permanently save this new book profile. Thus, the "newBookDetails" sequential file will now look like:

000_001 Village By The Sea Anita Desai Penguin Group Fiction 2012 01 01 available
000_002 Morning, Noon & Night Sidney Sheldon Scholastic Fiction 2012 01 01 available
000_003 Things Fall Apart Chinua Achebe Scribner Fiction 2012 01 01 available

The processes outlined above to show how book profiles are saved into the sequential file "newBookDetails", and how data form the sequential file is loaded into the program through the array list "bookDetails", also apply for the profiles of students. The only difference between them is the fact that the array list for student details is called "studentDetails", the sequential for the student profiles iis called "newStudentDetails", and the string variable that will be used to copy data from each profile in the array list into the sequential file is called "studentLine".

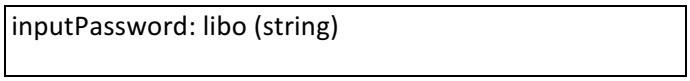
How the login details are stored

For the program to begin, and for the "studentDetails" and "bookDetails" file to be loaded into the program, the correct password and username combination have to be entered into the program before the program will start. For this reason, a correct password and username will already be assigned by the program, and librarian will just have to input the correct password and username combination. The correct password and username will be stored in individual variables, and every time the librarian inputs a password and username the will check to see if the inputted password and username match the correct password and username the program is supposed to receive, if they match then the program will go on to load the student and book details files; however, if they do not match, the program will not continue, and librarian will have to keep on inputting password and username combinations until the corr3ect combination is entered. The correct password and username will be stored in variables called "correctPassword" and "correctUsername" respectively; below is a representation of how the password and username will be stored:

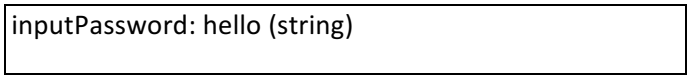
correctPassword : libo (string)

The diagram above shows that the correctPassword will only take string variables, and that the correct password that must be entered by the librarian is 'libo'. However, when the librarian inputs a password, this inputted password will be stored in a different variable – "inputPassword"; and after the password is inputted by the librarian, the program will then compare both passwords to check whether they match or not. For this, reason both variables will be made to store string values so that the program can easily compare the data in both variables. The diagram below is a representation of the

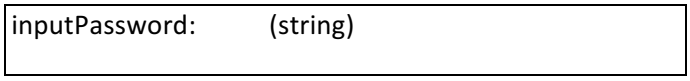
“inputPassword” variable, and the processes that go on when a password is being inputted by the librarian:



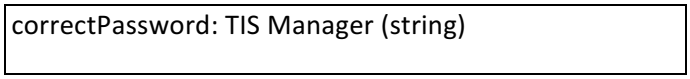
The diagram above shows that the correct password has been inputted by the librarian, and thus the program will go on to ask the librarian to input the correct username; however if the librarian does not enter the correct password, the data in the “inputPassword” variable will be erased, and the program will ask the librarian to input the correct password again. The diagram below shows this:



The diagram above shows that the wrong password has been inputted by the librarian since the correct password that must be inputted is “libo”; thus the data in the “inputPaassword” variable will be erased so that the correct password will be inputted by the librarian:



Just like for the password, the program will undergo the same processes when asking the user to input the correct username into the program. Therefore, the correct username will be stored in a variable called “correctUsername”, and the username the librarian inputs will be stored in a variable called “inputUsername”. The diagrams below are representations of the variables that will store the correct username, and the inputted username:



Just the period when the correct password had to be inputted by the librarian, the inputted username will also be compared with the correct username; and if they match, the program will go on to load the student and book details files, however, if they do not match then the librarian will have to keep on inputting usernames until the correct username is entered.

Algorithms

The following section will discuss all the methods that will be used by the program to be created:

Verifying the correct username is entered

This method will verify whether the username entered by the librarian is valid or not.

Pre-condition: The program does not ask the user to enter the correct password

Post-condition: The program goes on to ask the user to input the correct passsword

Pseudocode:

1. A variable is created that contains the correct username that must be entered
2. Loop created that will ask the user to input the correct username
3. Inputted username is compared to the correct username
4. If the inputted username matches the correct username, the program breaks out of the loop; and, if the inputted username does not match the correct username, the loop continues.

Verifying the correct password is entered

This method will verify whether the password entered by the librarian is valid or not.

Pre-condition: The username entered by the librarian has been validated to be correct

Post-condition: The student details and book details files are loaded, and the user is directed to the splash screen where the user chooses the action that he/she wants to undertake

Pseudocode:

1. A variable is created that contains the correct password that must be entered
2. A loop is created that will ask the user to input the correct password
3. The inputted password is compared to the correct password
4. If the inputted password matches the correct password, the program breaks out of the loop; and, if the inputted password does not match the correct password, the loop continues.

loadBooks()

This method loads the file that contains the details about each book stored in the library database into the program. This will make the program able to access the details of books that will be important for other methods to run. For example, when a book is being borrowed, the program will need to check whether that book has already been borrowed or not, and this information can only be found in the file that stores details about books.

Pre-condition: The array list for book details does not have the details of any book stored in it

Post-condition: The array list for book details has the details of books stored in it, and the program now has access to the details of books stored in the library

Pseudocode:

1. Open the file that stores the details of each book
2. Read the Id of the book, and store it in the array list for book details
3. Read the title of the book, and store it in the array list for book details
4. Read the author of the book, and store it in the array list for book details
5. Read the publisher of the book, and store it in the array list for book details
6. Read the genre of the book, and store it in the array list for book details
7. Read the condition of the book, and store it in the array list for book details
8. Read the year the book was stocked into the library, and store it in the array list for book details
9. Read the month the book was stocked into the library, and store it in the array list for book details
10. Read the day the book was stocked into the library, and store it in the array list for book details
11. Read the status of the book, and store it in the array list
12. Move to the next line and continue steps 2-12, until all the books in the file that stores book details have been read.

loadStudents()

This method loads the file that contains the details about each student who has borrowed a book before, into the program. This will make the program able to access the details of all students who have borrowed books before; this is important since this information must be accessed before other methods can run. For example, when a student is returning a borrowed book, the program will need to check whether the student has returned the book before its due date or not to be able to know whether to fine the student or nor, and this information can only be found in the file that stores the details about students.

Pre-condition: Only the file that contains details about books has been loaded into the program; also, the array list for student details has no data stored in it.

Post-condition: Both the file that contains the details about books and students have been loaded into the program; also the array list for student details has data contained in it.

Pseudocode:

1. Open the file that stores the details of each student
2. Read the Id of the student, and store it in the array list for student details
3. Read the first name of the student, and store it in the array list for student details
4. Read the last name of the student, and store it in the array list for student details

5. Read the grade of the student, and store it in the array list for student details
6. Read the Id of the book borrowed by the student, and store it in the array list for student details
7. Read the due year of the book borrowed by the student, and store it in the array list for student details
8. Read the due month of the book borrowed by the student, and store it in the array list for student details
9. Read the due day of the book borrowed by the student, and store it in the array list for student details
10. Read the year the student returned the book he/she borrowed, and store it in the array list for student details
11. Read the month the student returned the book he/she borrowed, and store it in the array list for student details
12. Read the day the student returned the book he/she borrowed, and store it in the array list for student details
13. Read the total fine accumulated by the student, and store it in the array list for student details
14. Move to the next line and continue steps 2-12, until all the students in the file that stores student details have been read.

writeStudentsToFile()

This method saves all the details about students in the array list for students into a file that will store the details about these students.

Pre-condition: Data in the array list for students have not been stored in the file that stores the details about students

Post-condition: The data in the array list for students have been written in the file that stores the details about students

Pseudocode:

1. Create a string to store the details of each student in the array list for students. This variable will be called “studentLine”
2. Open loop
3. Read the Id of the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
4. Read the first name of the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
5. Read the last name of the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
6. Read the grade of the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
7. Read the Id of the book borrowed by the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”

8. Read the due year of the book borrowed by the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
9. Read the due month of the book borrowed by the student ffrom the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
10. Read the due day of the book borrowed by the student from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
11. Read the year the student returned the book he/she borrowed from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
12. Read the month the student returned the book he/she borrowed from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
13. Read the day the student returned the book he/she borrowed from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
14. Read the total fine the student has accumulated from the array list for student details, store it in “studentLine” and add a tab character to “studentLine”
15. Output “studentLine” into the file that will store the details of each student
16. Move to the next student in the array list for student details, and repeat steps 3-14
17. Output “studentLine” in the next line in the file that will store the details of each student
18. Repeat steps 16-17 until the details of every student in the array list for student details have been stored in the file that will store the details of each student

writeBooksToFile()

This method saves all the details about books in the array list for books into a file that will store the details about these books.

Pre-condition: Data in the array list for books have not been stored in the file that stores the details about books

Post-condition: The data in the array list for books have been written in the file that stores the details about books

Pseudocode:

1. Create a string to store the details of each book in the array list for books. This variable will be called “bookLine”
2. Open loop
3. Read the Id of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
4. Read the title of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
5. Read the author of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
6. Read the publisher of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”

7. Read the genre of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
8. Read the condition of the book from the array list for book details, store it in “bookLine” and add a tab to “bookLine”
9. Read the year the book was stocked into the library from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
10. Read the month the book was stocked into the library from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
11. Read the day the book was stocked into the library from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
12. Read the status of the book from the array list for book details, store it in “bookLine” and add a tab character to “bookLine”
13. Output “bookLine” into the file that will store the details of each book
14. Move to the next book in the array list for book details, and repeat steps 3-12
15. Output “bookLine” in the next line in the file that will store the details of each book
16. Repeat steps 14-15 until the details of every book in the array list for book details have been stored in the file that will store the details of each book

addBook()

This method will be used during the re-stocking stage to the details of new books to the library database.

Pre-condition: The details of the new books are not included in the sequential file that will store the details of each book

Post-condition: The sequential file contains the details of the new book

Pseudocode:

1. The user enters the Id of the new book being added to library database
2. The program searches through the book details array to check if any book in the array for book details contains the same Id as the inputted Id
3. If no book is found to have the same Id as the inputted Id, the program goes on to ask the librarian to enter the title of the new book.
4. Then the program goes on to ask the librarian to input the title of the new book
5. The program asks the librarian to input the author of the new book
6. The program asks the librarian to input the year the new book was stocked into the library
7. The program asks the librarian to input the month the new book was stocked into the library
8. The program asks the librarian to input the day the new book was stocked into the library
9. The program asks the librarian to input the genre of the new book
10. The program asks the librarian to input the publisher of the new book
11. The program asks the librarian to input the status of the new book
12. The details of the new book are then saved into the sequential file that will store book details

searchBook(String bookId)

This is the method that will be used by the addBook() method to search through the array list for book details to find out whether the inputted Id in the addBook() method already exists in the array list for book details.

Parameters: the Id of the new book to be added to the library database

Returns: returns a -1 if the inputted book Id does not exist in the library database, or returns the index of the book if the inputted Id is found in the array list for book details

Pre-condition: the details of the new book cannot be added to the library database

Post-condition: If the inputted Id is not found in the library database, then the details of the new book can be added to the library database; however, if the inputted Id is found in the library database, then the details of the new book cannot be added to the library database

Pseudocode:

1. Open loop
2. Search through the book Id's of each book in the array list for book details.
3. If the inputted book Id is found in the array list for book details, then return the index of that specific book; and break out of the loop.
4. If the whole book details array list has been searched and the inputted Id is not found in the array list for book details, then return a -1.

addStudent()

This method will be used during the stage when a book is being borrowed from the library, and the Id of the student who intends to borrow the book is not found in the library database.

Pre-condition: The profile of the student has not been added to the sequential file that stores the profiles of students

Post-condition: The profile of the student has been added to the sequential file that stores the profiles of students

Pseudocode:

1. The user enters the Id of the new student profile to be added
2. The program searches through the array list for student details to find out whether a student profile with the same Id exists or not

3. If the inputted Id is not found in the array list for student details, then the program asks the user to input the first name of the new student profile
4. The program asks the user to input the last name of the new student profile
5. The program asks the user to input the grade of the new student profile
6. The book borrowed by the student is set to the book that he/she intended to borrow at the beginning of the borrowing stage
7. The program asks the user to input the due year of the book being borrowed by the student
8. The program asks the user to input the due month of the book being borrowed by the student
9. The program asks the user to input the due day of the book being borrowed by the student
10. The details of the new student profile are added to the sequential file that stores the details of students
11. The status of the borrowed book is set to "unavailable" to show that the borrowed book is out of the library

searchStudent(String studentId)

This method will be used during the stage a book is being borrowed from the library, and during the stage a new profile is being added to the library database; this method will search through the array list for student details to find out whether an inputted student Id exists in the library database or not.

Parameter: a student Id

Return: Returns a -1 if the student Id does not exist in the array list for student details; however, if the student Id is found in the array list for student details, it returns the index of that student profile

Pre-condition: The program does not know whether the inputted Id exists in the library database or not

Post-condition: The user and the program know whether the inputted Id exists in the library database or not

Pseudocode:

1. Open loop
2. Search through the student Id's of each student in the array list for student details.
3. If the inputted student Id is found in the array list for student details, then return the index of that specific student; and break out of the loop.
4. If the whole student details array list has been searched and the inputted Id is not found in the array list for student details, then return a -1.

borrowBook()

This method will be used during the stage a book is being borrowed from the library; this is the method that will append the Id of the book being borrowed to the student's profile, and will also set the due date of the book being borrowed from the library.

Pre-condition: The sections of the student profile that stores the due date and Id of the borrowed book are empty

Post-condition: The sections of the student profile that stores the due date and Id of the borrowed book have content in them

Pseudocode:

1. The user inputs the Id of the book to be borrowed
2. The program runs the searchBook(String Id) method to find out whether the inputted book Id exists in the library database or not
3. The program checks whether the status of the book to be borrowed is 'available' or 'unavailable'
4. If the status of the book is 'available', then the program asks the user to input the Id of the student who intends to borrow the program
5. The program runs the searchStudent(String studentId) method to find out whether the inputted student Id exists in the array list for student details or not
6. If the inputted Id is found not to exist in the library database, the program then runs the addStudent() method
7. However, if the inputted studentId is found to exist in the library database, then the program sets the section of the student profile that stores the Id of the book borrowed by the student to the book Id inputted by the user in step 1
8. The program asks the user to input the due year of the book to be borrowed
9. The program asks the user to input the due month of the book to be borrowed
10. The program asks the user to input the due day of the book to be borrowed
11. The program then stores all the information inputted into the student profile into the sequential file that stores the details of students
12. The status of the borrowed book is set to 'unavailable' (this is to show that the borrowed book is no longer in the library), and this is stored in the sequential file that will store the details of books.

returnBook()

This method will be used during the stage a borrowed book is being returned by a student; this method will append data to the sections of the student profile that store the return dates of books return by students, and will also append data to the section of the student profile that will store information o the fines that students have accumulated.

Pre-condition: The sections of the student profile that store the due date and Id of the book borrowed by the student have data in them, whilst the sections of the student profile that store the return date of the book returned by the student have no data in them; also the section of the student profile that stores the total fine accumulated by the student has not been updated. Finally, the section of the book profile that stores the status of the borrowed book remains as 'unavailable'.

Post-condition: The sections of the student profile that store the return date of the book returned by the student have data in them whilst the section of the student profile that stores the total fine accumulated by the student has been updated. The data that was previously in the sections of the student profile that store the due date and Id of the book borrowed by the student, have been erased. Finally, the status of the returned has been set to 'available' to show that the book is now in the library, and can be borrowed.

Pseudocode:

1. The program asks the user to input the Id of the book to be borrowed
2. The program runs the searchBook(String bookId) method to check whether the inputted book Id exists in the library database or not
3. If the inputted book Id has been found to exist in the library database, then the program goes on to check whether status of the book whose Id has been inputted into the program is 'available' or 'unavailable'
4. If the status of the book is 'unavailable', then the program then asks the user to input the Id of the student returning the book
5. The program then runs the searchStudent(String student) method to check whether the inputted student Id exists in the library database or not
6. If the student Id is found to be in the library database, then the program checks whether the data in the section of the student profile that stores the Id of the book borrowed by the student, matches the book Id inputted in step 1
7. If they match, then the program goes on to ask the user to input the year the book was returned by the student
8. Next, the program asks the user to input the month the book was returned by the student
9. Next, the program asks the user to input the day the book was returned by the student
10. The program then checks whether the book was returned after its due date
11. If the student is found to return the book after its due date, then the program add 1 to the section of the student profile that stores the total fine accumulated by the student
12. The program erases all the data in the section of the student profile that stores the due date and Id of the book borrowed by the student; this is to show that the student no longer has the returned book in his/her possession
13. Next all the updated data to the student profile is stored in the sequential file that stores the details of students
14. The status of the returned book is set to 'available' (this is to show that the book is now in the library), and this update to the book's profile is stored in the sequential file that stores the details of books.

searchBookByAuthor()

This method will be used during the stage the librarian needs to create lists of a specific category of books in the library; this method looks for books in the library database that were written by a specific author, and outputs the details of these books in the form of a list.

Pre-condition: There is no output of the list of books written by the specific author

Post-condition: A list showing the details of books written by a specific author has been outputted by the program

Pseudocode:

1. The program asks the user to input the name of the author he/she wants to base the search on
2. Open loop
3. The program looks through the section of the array list for book details that stores the author of each book, and checks whether the data there matches the inputted author in step 1
4. If the authors are found to match, the program outputs the author, title, genre, publisher, condition, status, and Id of that specific book.
5. Steps 3- 4 are repeated till all the books in the array list for book details have been looked through.

searchBookByPublisher()

This method will also be used during the stage the librarian needs to create lists of a specific category of books in the library; this method looks for books in the library database that were published by a specific publisher, and outputs the details of these books in the form of a list.

Pre-condition: There is no output of the list of books published by the specific publisher

Post-condition: A list showing the details of books published by a specific publisher has been outputted by the program

Pseudocode:

1. The program asks the user to input the name of the publisher he/she wants to base the search on
2. Open loop
3. The program looks through the section of the array list for book details that stores the publisher of each book, and checks whether the data there matches the inputted publisher in step 1
4. If the publishers are found to match, the program outputs the publisher, title, author, genre, condition, status, and Id of that specific book.
5. Steps 3- 4 are repeated till all the books in the array list for book details have been looked through.

searchBookByGenre()

Just like the two methods described above, this method will also be used during the stage the librarian needs to create lists of a specific category of books in the library; this method looks for books in the library database that are of a specific genre, and outputs the details of these books in the form of a list.

Pre-condition: There is no output of the list of books of the specific genre

Post-condition: A list showing the details of books of the specific genre has been outputted by the program

Pseudocode

1. The program asks the user to input the genre he/she wants to base the search on
2. Open loop
3. The program looks through the section of the array list for book details that stores the genre of each book, and checks whether the data there matches the inputted genre in step 1
4. If the genres are found to match, the program outputs the genre, title, author, publisher, condition, status, and Id of that specific book.
5. Steps 3- 4 are repeated till all the books in the array list for book details have been looked through.

searchStudentByFine()

This method searches through the array list for student details and outputs the details of all the students who have been fined by the library; this method will be used when the library needs to create a list of students who have been fined, in order to present this list to the accounts office.

Pre-condition: No list of students who have been fined by the library has been outputted

Post-condition: A list of students who have been fined by the library has been outputted

Pseudocode:

1. Open loop
2. The program looks through the profiles of each student in the array list for student details, and checks whether the number stored in the total fine section for each student is greater than 0
3. If the number in the total fine section is greater than 0, then the Id, first name, last name, grade, and total fine accumulated by the student is outputted.
4. The program repeats steps 2-3 until all the student profiles in the array list for student details have been looked through.

Modular Organization

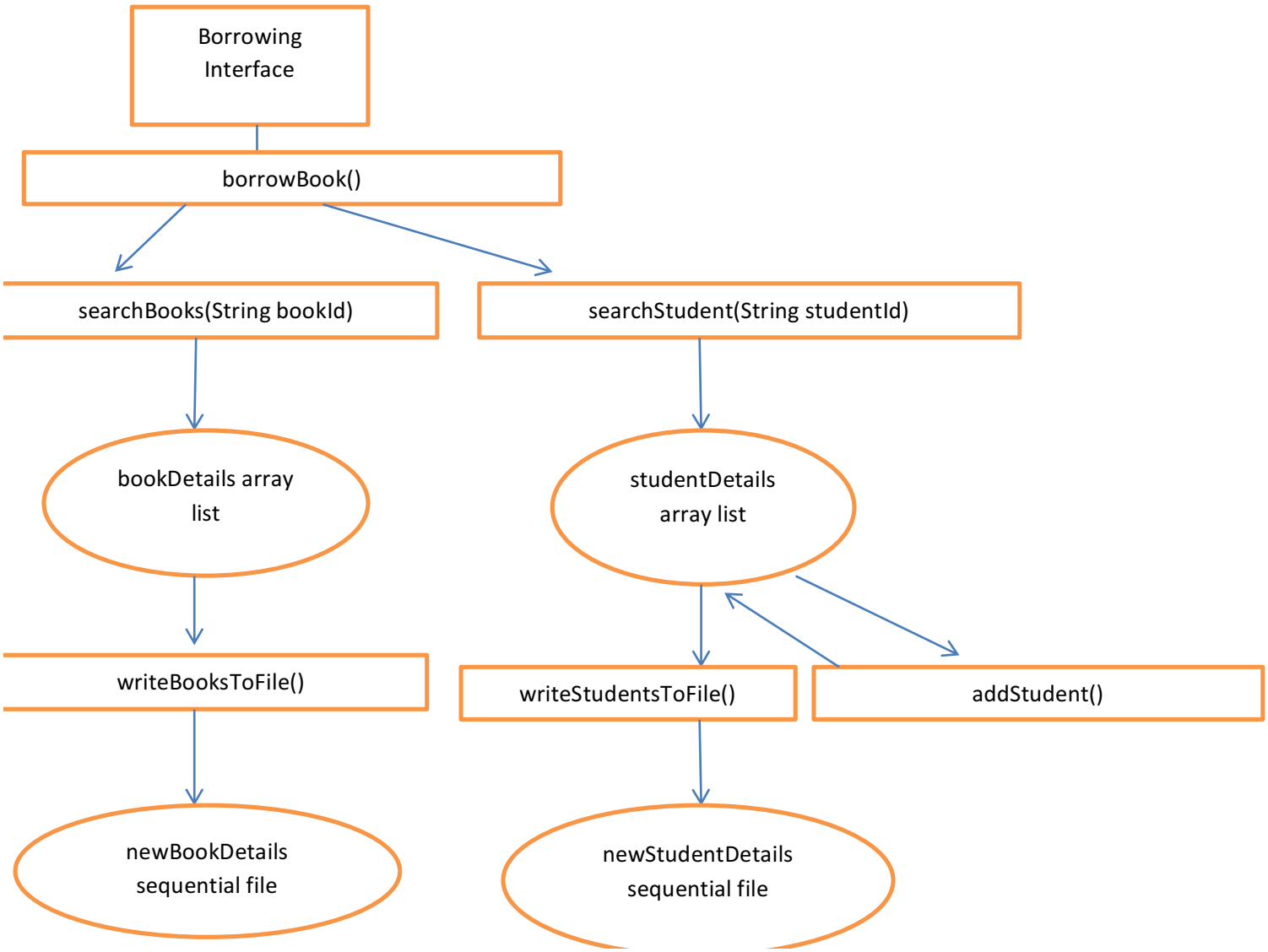
The program performs four main functions, and these are:

1. The program allows books to be borrowed from the library
2. The program allows borrowed books to be returned to the library
3. The program allows new books to be stocked into the library
4. The program allows the librarian to search for books of a particular category, and to search for students who have been fined.

For this reason, the program been divided into four main classes, and these are the borrowing interface, the returning interface, the re-stocking interface, and the searching interface.

In the following section, I will discuss and describe each of the four main classes.

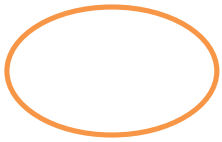
The Borrowing Interface



Key

- Class

- Method



-Data Structure

The diagram above shows how the borrowing interface class is organized.

This diagram illustrates that once the user enters the borrowing interface, the program begins to run the borrowBook() method. Once the borrowBook() method begins, the program asks the user to input the Id of the book to be borrowed.

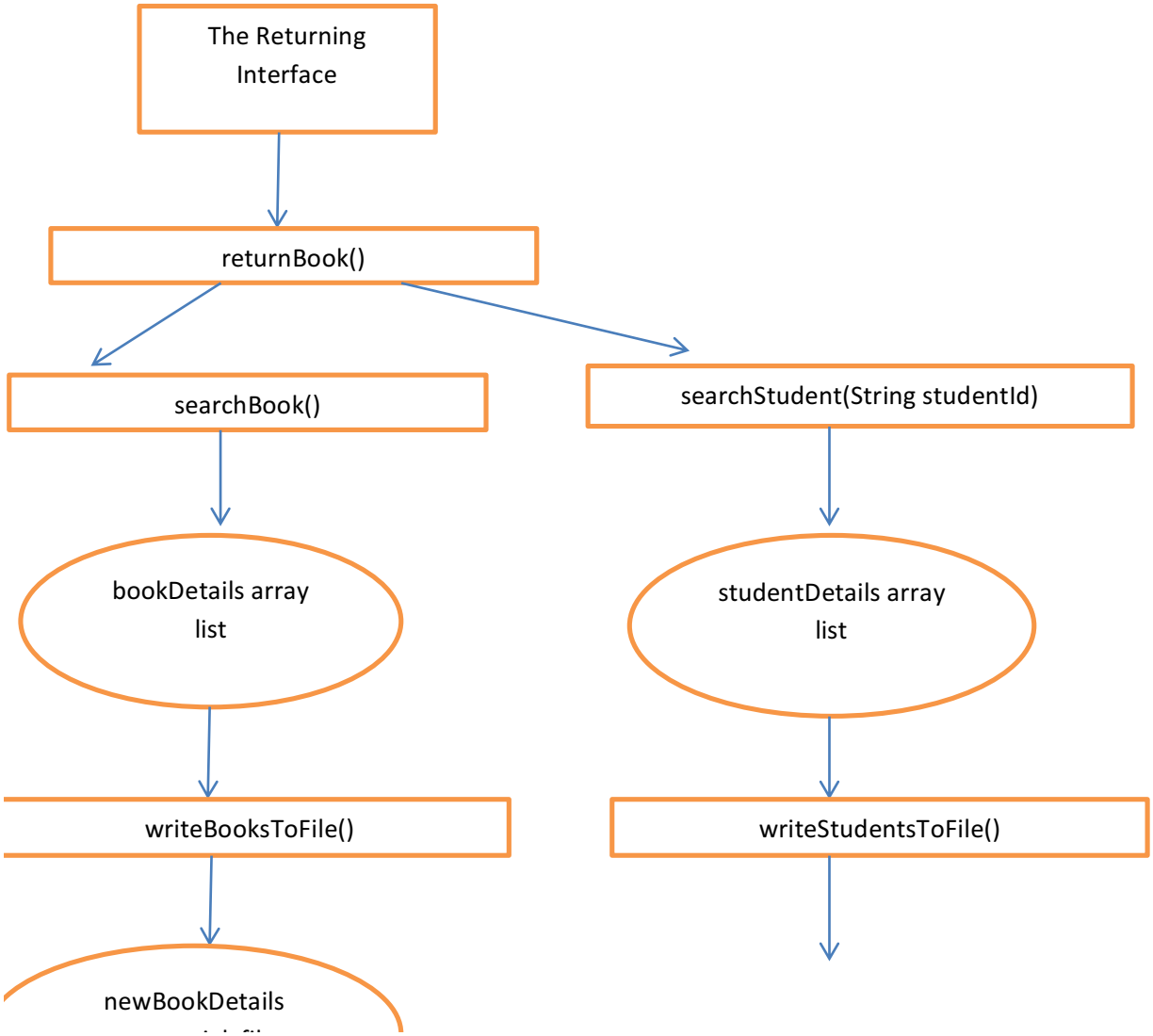
After the user inputs the Id of the book to be borrowed, the searchBook(String bookId) method is run, in order to check whether the inputted book Id exists. If the inputted book Id is found to exist in the bookDetails array list, the program goes on to check whether the book's status say 'available'; if the book's status says 'available', the program then asks the user to input the Id of the student who intends to borrow the book.

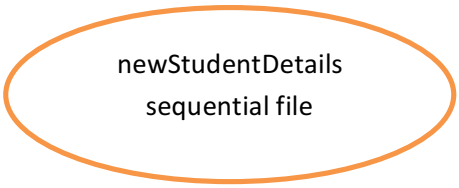
Once the Id of student is inputted, the searchStudent(String studentId) method is run in order to check whether the inputted student Id exists I the studentDetails array list.

If the student Id is found in the studentDetails array list, the program then checks whether the student already has a book borrowed- this is because, as stated in the criteria for success, students will be allowed to borrow only one book. Thus if the student whose Id was inputted earlier is found not to have any book in his/her possession, then the program allows the librarian to update the profile of the student to show that the student has just borrowed a book. This update to the student's profile is sent to the studentDetails array list, and the writeStudentToFile() method is run to save the changes made to the studentDetails array list in the newStudentDetails sequential file.

However, if the student Id inputted earlier is found not to exist in the studentDetails array list, the addStudent() method is run. During the addStudent() method, the details of the new student profile are inputted, and this includes the Id of the book the student intends to borrow, and the due date of the borrowed book. After this, the new student profile is added to the studentDetails array list; from there, the writteStudentsToFile() method is run to save the changes made to the studentDetails array list in the newStudentDetails sequential file.

The Returning Interface





The returning interface class begins with the running of the returnBook() method; during the running of this method, the program asks the user to input the Id of the book to be returned.

After the user inputs the Id of the book to be returned, the searchBook(String bookId) method is run. This method checks whether the inputted book Id exists in the bookDetails array list; if the inputted book Id exists in the bookDetails array list, the program checks whether the status of the book says "unavailable". If the status of the book says "unavailable", then it means that the book is out of the library, and thus it can be returned; however, if the status if the book reads "available", then it means that the book is already in the library and thus an error message is displayed to show that the book cannot be returned since it is already in the library – a book cannot be returned if it was not borrowed.

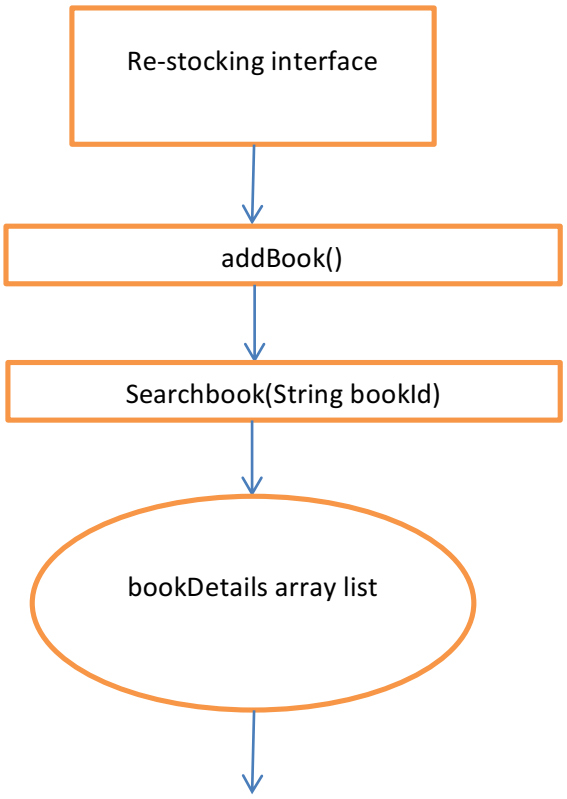
If the status of the book whose Id was inputted reads "unavailable", then the program goes on to ask the user to input the Id of the student returning the book. Next, the program runs the searchStudent(String student Id) method to check if the inputted student Id exists in the studentDetails array list.

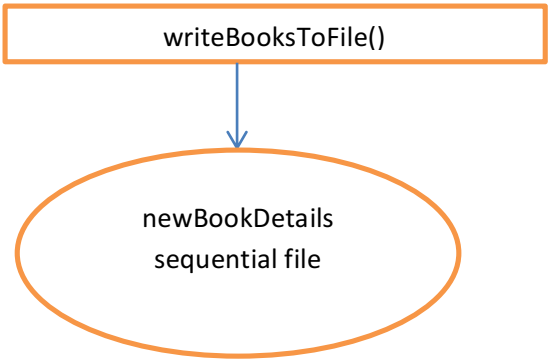
If the student Id is found to exist in the studentDetails array list, the program goes on to check whether the bookBorrowed section of that student profile contains the same book Id as the book Id the librarian inputted at the beginning of the returning stage. If these Id's are found to be the same, then the user will now be able to update the profile of the student, this includes inputting the date the book was returned by the student.

After this, the update to the student's profile is made in the studentDetails array list, and then these changes made to the studentDetails array list are saved in the newStudentDetails sequential file, using the writeStudentsToFile() method.

Also the status if the returned book is changed from "unavailable" to "available" in the bookDetails array list. Finally, the writeBooksToFile() method is used to save the changes made to the bookDetails array list in the newBookDetails sequential file.

The Re-stocking Interface





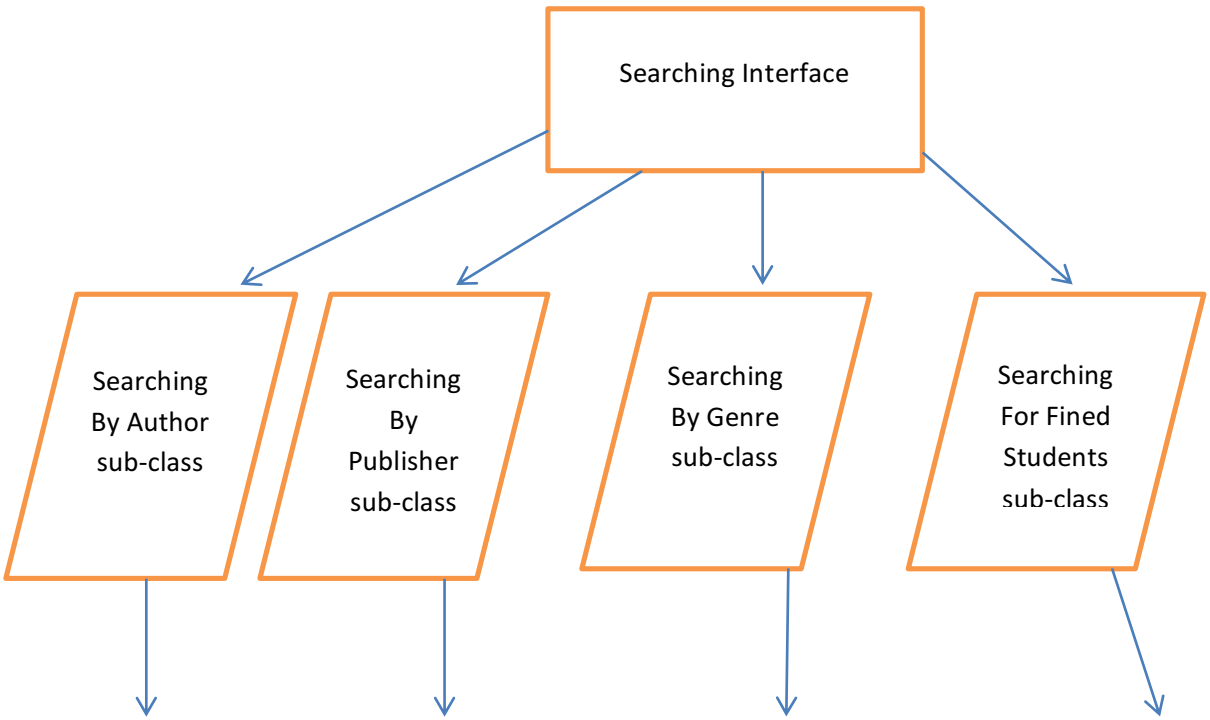
During the stage a new book is being added to the library database, the program sends the user to the re-stocking interface.

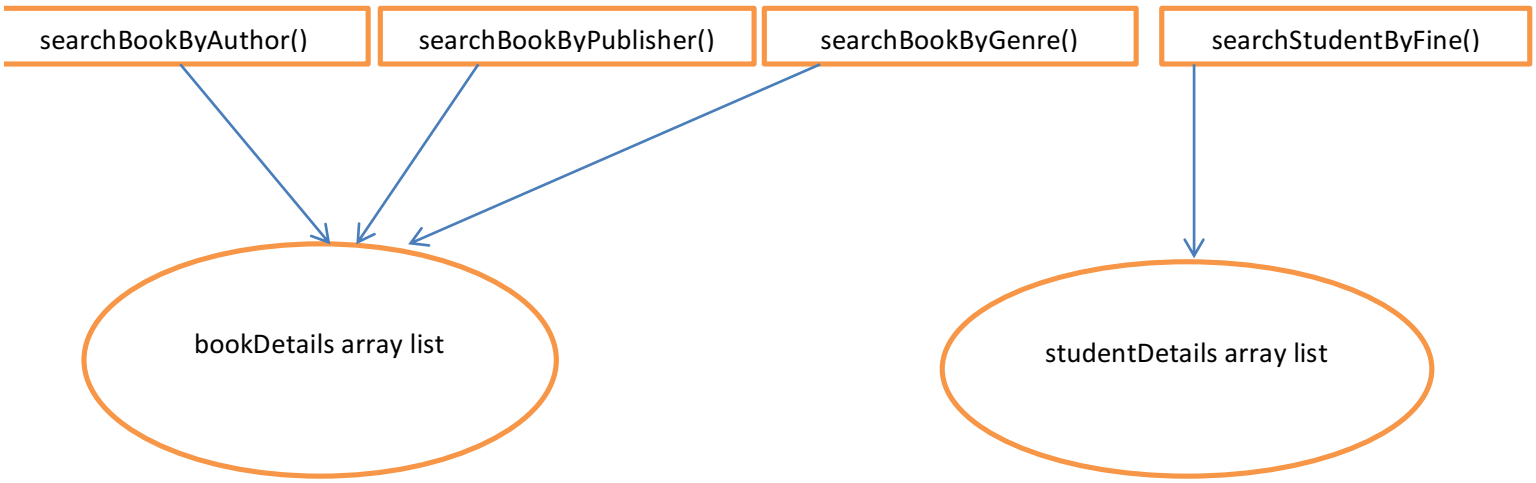
In the re-stocking interface, the program begins by running the addBook() method. At the beginning of the addBook() method, the program asks the librarian to input the Id of the new book to be added to the library database.

After the user input the Id of the new book, the searchBook(String bookId) is run to check if the inputted book Id exists in the bookDetails array list. If the book Id is found to be non-existent in the bookDetails array list, the user is then asked to input the details of the new book.

Next, the details of the new book profile are added to the bookDetails array list. Then, using the writeBooksToFile() method, the changes made to the bookDetails array list are saved in the newBookDetails sequential file.

The Searching Interface





As the diagram above shows, the searching interface is divided into four subclasses; the section below will describe each of these subclasses.

The Searching By Author sub-class

In this class, the program searches through the library database for books written by a specific author.

This class begins by running the searchBookByAuthor() method which starts by asking the librarian to input the name of the author the search is to be based on. After the name of the author has been by the librarian, the program searches through the bookDetails array list to find books whose author section have the same author as the author the librarian inputted earlier.

At the end of the searchBookByAuthor() method, the program outputs all the books in the bookDetails array list that were written by the author searched for.

The Searching By Publisher Interface

In this class, the program runs the searchBookByPublisher() method.

In this method, the program first asks the user to input the name of the publisher the search is to be based on. After the user inputs the name of the publisher, the program searches through the bookDetails array list for books whose publisher section contain the same publisher as the publisher inputted by the user.

By the end of the searchByPublisher() method, the program should have outputted all the books published by the inputted publisher.

The Searching By Genre Interface

In this class, the program searches through the library database and outputs a list of all the books of an inputted genre.

This class runs the searchBookByGenre() method which begins by asking the user to input the genre the search is to be based on. Information obtained from the problem analysis show that the only genres that can be in the library are “fiction” and “non-fiction”. Thus the only inputs this method will accept from the user are either “fiction” or “non-fiction”.

After the user inputs genre the search is to be based on, the program searches through the bookDetails array list to find book profiles whose genre section contain the same genre as the inputted genre.

After the whole bookDetails array list has been lookd through, the program outputs all the books in the library database of the same genre as the inputted genre.

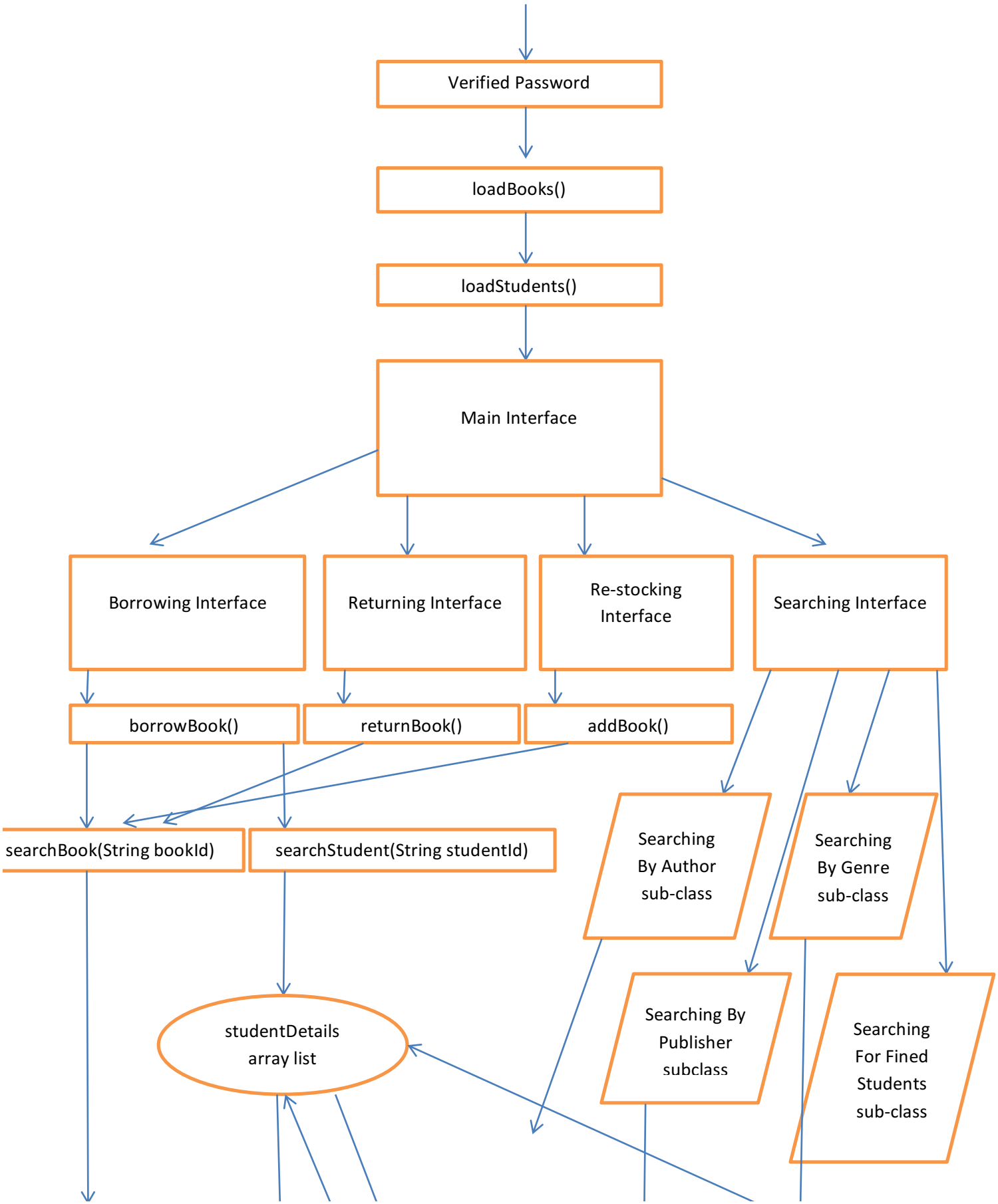
The Searching For Fined Students Interface

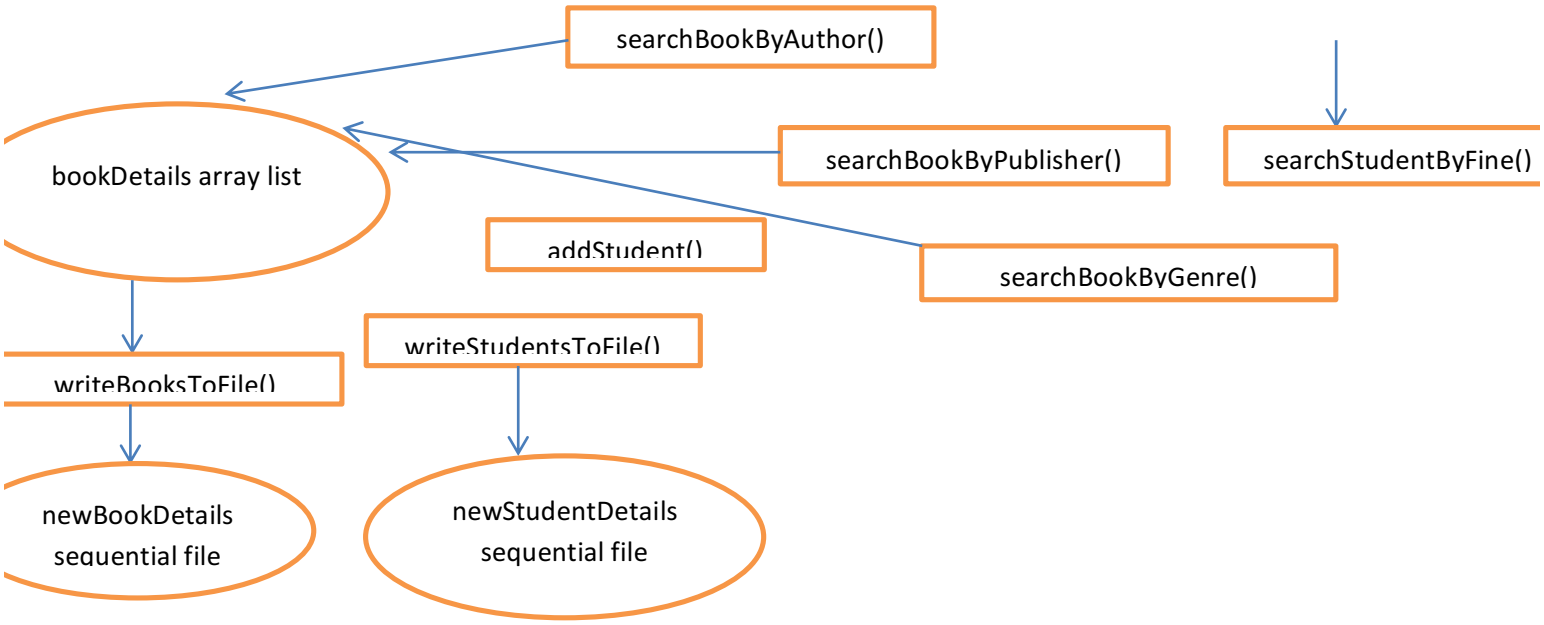
In this class, the program searches through the library database and outputs the details of all students who have been fined by the library.

This class runs the searchStudentByFine() method which searches through the studentDetails array list for students whose fine sections have numbers greater than zero stored in them.

After the program searches through the whole studentDetails array list, it outputs all the students in the studentDetails array list who have accumulated fines.

The above sections have shown how each of the individual classes in the program are organized; thus when all the classes are put together, the modular organization of the whole program becomes:


Verified Password
loadBooks()
loadStudents()
Main Interface
Borrowing Interface
Returning Interface
Re-stocking Interface
Searching Interface
borrowBook()
returnBook()
addBook()
searchBook(String bookId)
searchStudent(String studentId)
Searching By Author sub-class
Searching By Genre sub-class
Searching By Publisher subclass
Searching For Fined Students sub-class
studentDetails array list



As the modular diagram of the whole program shows, the username and password inputted by the librarian must be verified before loadStudents() and loadBooks() methods will run. This is to ensure that only the librarian will be able to access the program since only the librarian will know the correct username and password.

The loadStudents() and loadBooks () methods will be used by the program to put all the changes that were made to the library database during the last session into the studentDetails and bookDetails array lists respectively. This will make it possible for the user to add new data to the library database and use the already saved data in library database once the program enters into the main interface.

Mastery Factors

Below is a table showing the mastery factors the solution will achieve, and the methods that will be used to achieve each respective mastery factor.

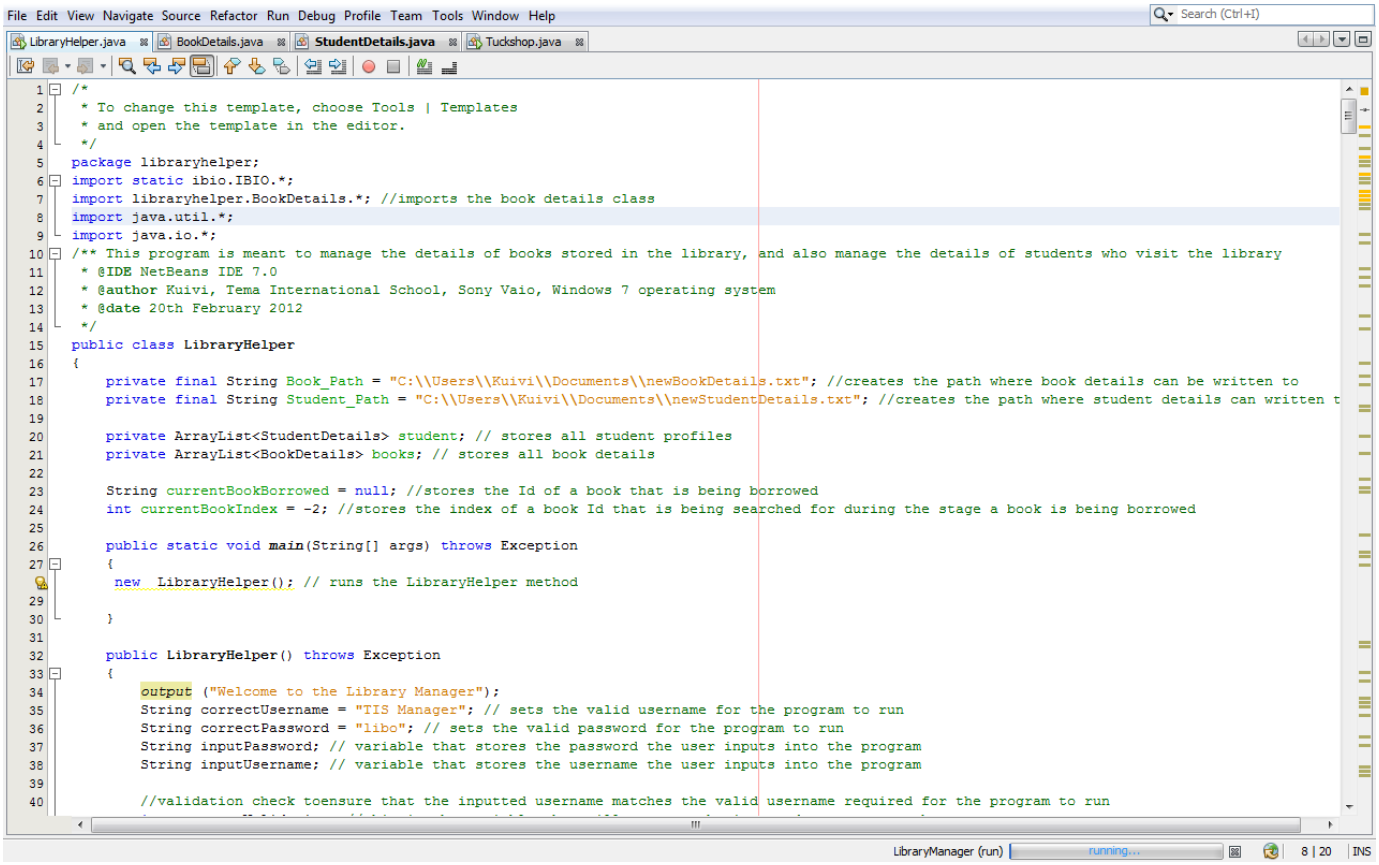
1. Arrays	studentDetails and bookDetails will make use of arrays
2. User-defined objects	Student profiles and book profiles will be stored in theStudentDetails and BookDetails objects
3. Objects as data records	The record of each student and book will be stored in the StudentDetails and BookDetails objects
4. Simple Selection (if-else)	The mutator methods in the BookDetails and

	StudentDetails classes will make use of thiis, to validate data inputted by the user
5. Complex selection	The borrowBook() and returnBook() methods will make use of this
6. Loops	The searchBook(String Id) method will make use of loops; and also the borrowBook() and returnBook() methods will use loopsto allow the uuse to continue borrowing or returning books until there are no more books to return or borrow. The addBook() method will also use loops to allow the librarian to continue adding new book profiles until there are no more profiles to add
7. User-defined methods	The borrowBook(), addBook(), addStudent(), searchBookByAuthor(), serachBookBy Publisher, searchBookByGenre(), and searchStudentByFine are alll user-defined methods
8. User-defined methods with parameters	The searchBook(String bookId) and searchStudent(String studentId) methods are user-defined methods that will be used fo find out whether the inputted Id's exist in the library database
9. User-defined methods with appropriate return values	The searchBook(String bookId) and searchStudent(String studentId) methods will return integers representing the index of a profile if the profile exists, or will return -1 if the profile does not exist
10. Searching	The searchBook(String bookId), searchStudent(String studentId), searchBookByAuthor(), searchBookByPublisher(), searchBookByGenre(), and searchStudentByFine methods will search through the studentDetails and bookDetails array lists
11. File i/o	The writeStudentsToFile(), writeBooksToFile(), loadStudents(), and loadBooks() methods will read and write from the newStudentDetails and newBookDetails files

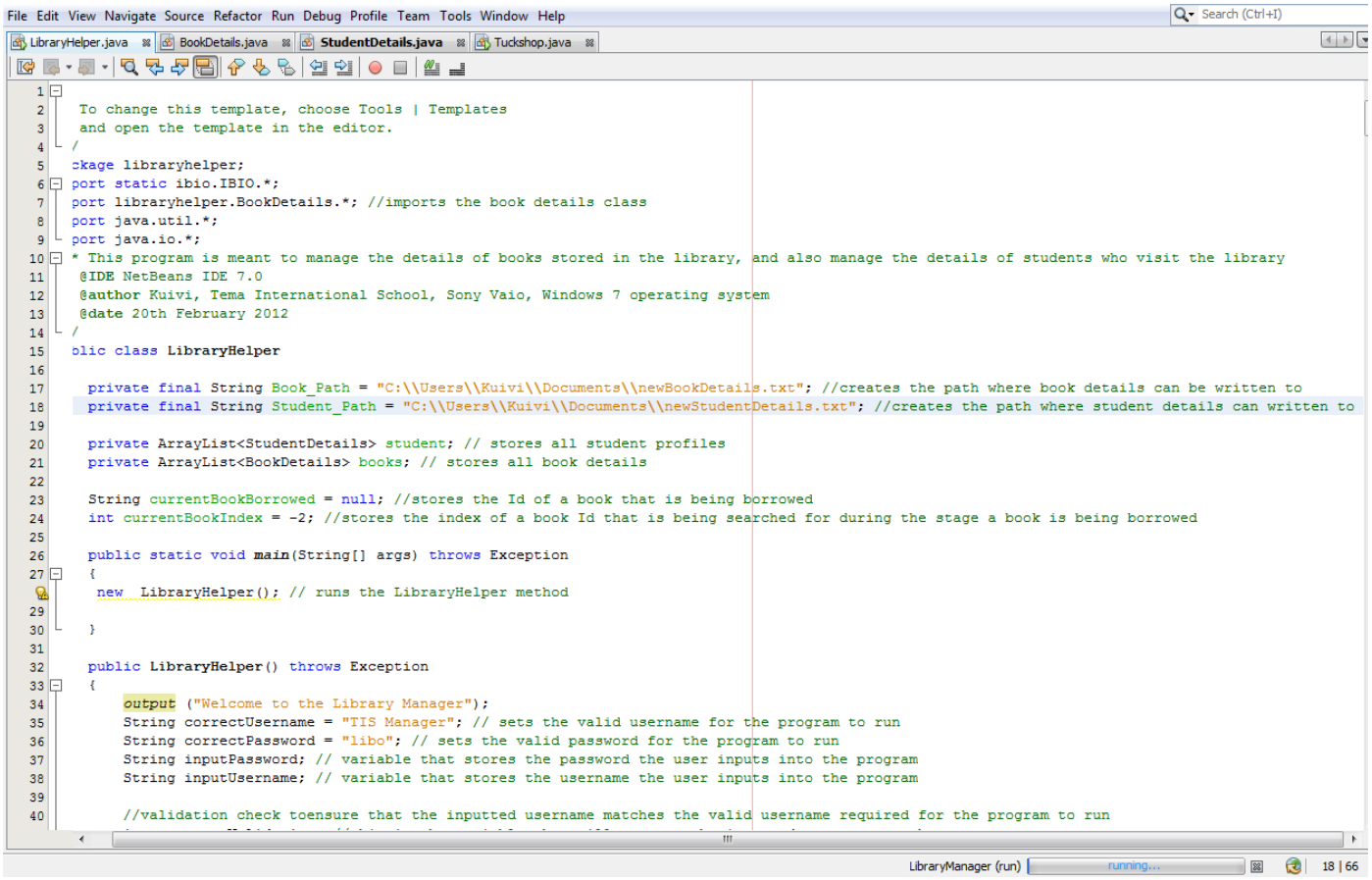
Programming

Main Class

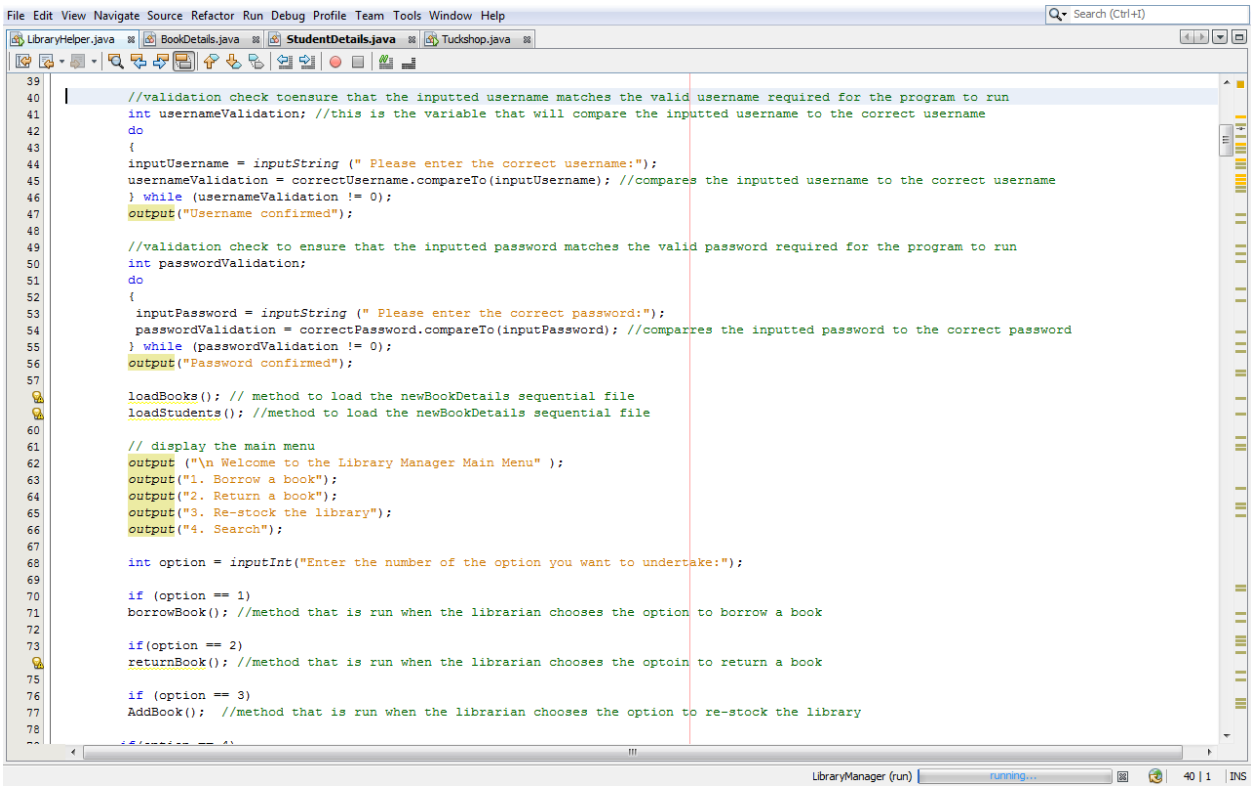
1.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package libraryhelper;
import static ibio.IBIO.*;
import libraryhelper.BookDetails.*; //imports the book details class
import java.util.*;
import java.io.*;
/** This program is meant to manage the details of books stored in the library, and also manage the details of students who visit the library
* @IDE NetBeans IDE 7.0
* @author Kuivi, Tema International School, Sony Vaio, Windows 7 operating system
* @date 20th February 2012
*/
public class LibraryHelper
{
private final String Book_Path = "C:\\Users\\Kuivi\\Documents\\newBookDetails.txt"; //creates the path where book details can be written to
private final String Student_Path = "C:\\Users\\Kuivi\\Documents\\newStudentDetails.txt"; //creates the path where student details can written t
private ArrayList<StudentDetails> student; // stores all student profiles
private ArrayList<BookDetails> books; // stores all book details
String currentBookBorrowed = null; //stores the Id of a book that is being borrowed
int currentBookIndex = -2; //stores the index of a book Id that is being searched for during the stage a book is being borrowed
public static void main(String[] args) throws Exception
{
new LibraryHelper(); // runs the LibraryHelper method
}
public LibraryHelper() throws Exception
{
output ("Welcome to the Library Manager");
String correctUsername = "TIS Manager"; // sets the valid username for the program to run
String correctPassword = "libo"; // sets the valid password for the program to run
String inputPassword; // variable that stores the password the user inputs into the program
String inputUsername; // variable that stores the username the user inputs into the program
//validation check toensure that the inputted username matches the valid username required for the program to run
LibraryManager (run)
running...
8 | 20
INS

2.

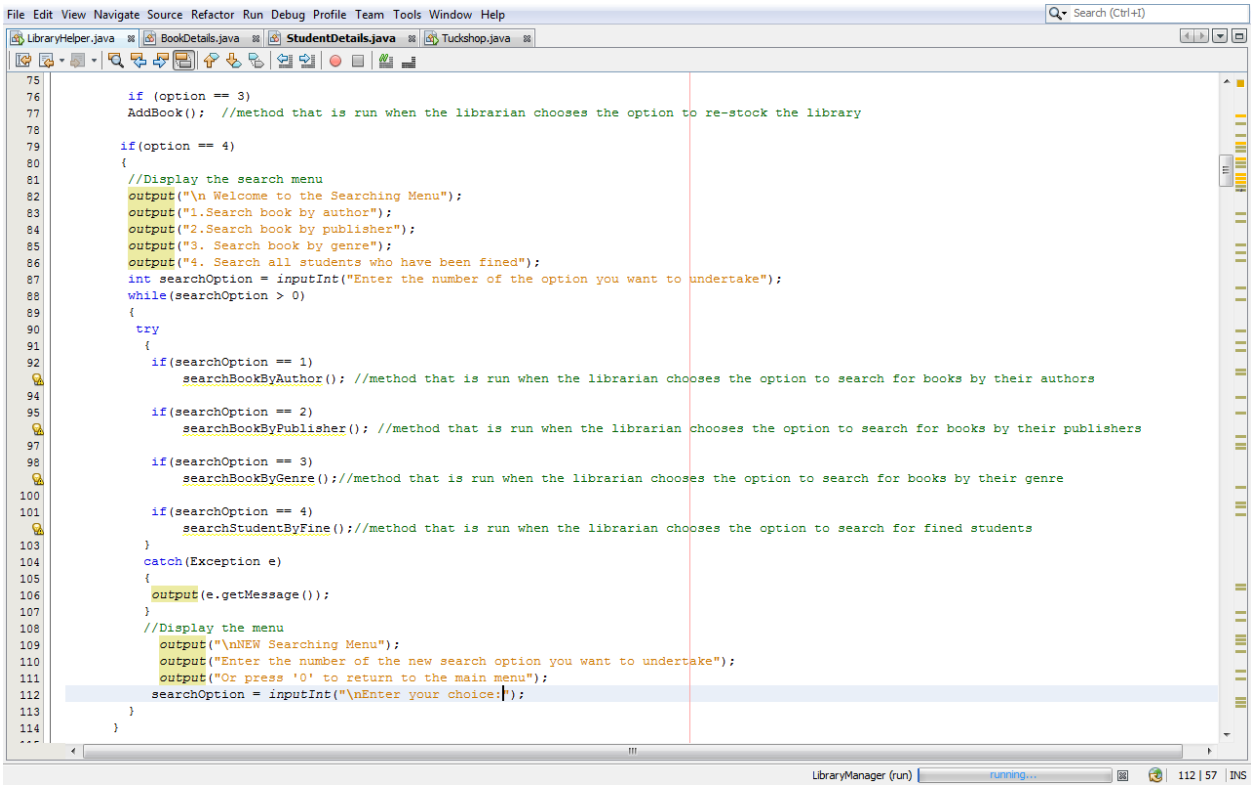

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
To change this template, choose Tools | Templates
and open the template in the editor.
ckage libraryhelper;
port static ibio.IBIO.*;
port libraryhelper.BookDetails.*; //imports the book details class
port java.util.*;
port java.io.*;
* This program is meant to manage the details of books stored in the library, and also manage the details of students who visit the library
@IDE NetBeans IDE 7.0
@author Kuivi, Tema International School, Sony Vaio, Windows 7 operating system
@date 20th February 2012
blic class LibraryHelper
private final String Book_Path = "C:\\Users\\Kuivi\\Documents\\newBookDetails.txt"; //creates the path where book details can be written to
private final String Student_Path = "C:\\Users\\Kuivi\\Documents\\newStudentDetails.txt"; //creates the path where student details can written to
private ArrayList<StudentDetails> student; // stores all student profiles
private ArrayList<BookDetails> books; // stores all book details
String currentBookBorrowed = null; //stores the Id of a book that is being borrowed
int currentBookIndex = -2; //stores the index of a book Id that is being searched for during the stage a book is being borrowed
public static void main(String[] args) throws Exception
{
new LibraryHelper(); // runs the LibraryHelper method
}
public LibraryHelper() throws Exception
{
output ("Welcome to the Library Manager");
String correctUsername = "TIS Manager"; // sets the valid username for the program to run
String correctPassword = "libo"; // sets the valid password for the program to run
String inputPassword; // variable that stores the password the user inputs into the program
String inputUsername; // variable that stores the username the user inputs into the program
//validation check toensure that the inputted username matches the valid username required for the program to run
LibraryManager (run)
running...
18 | 66

3.



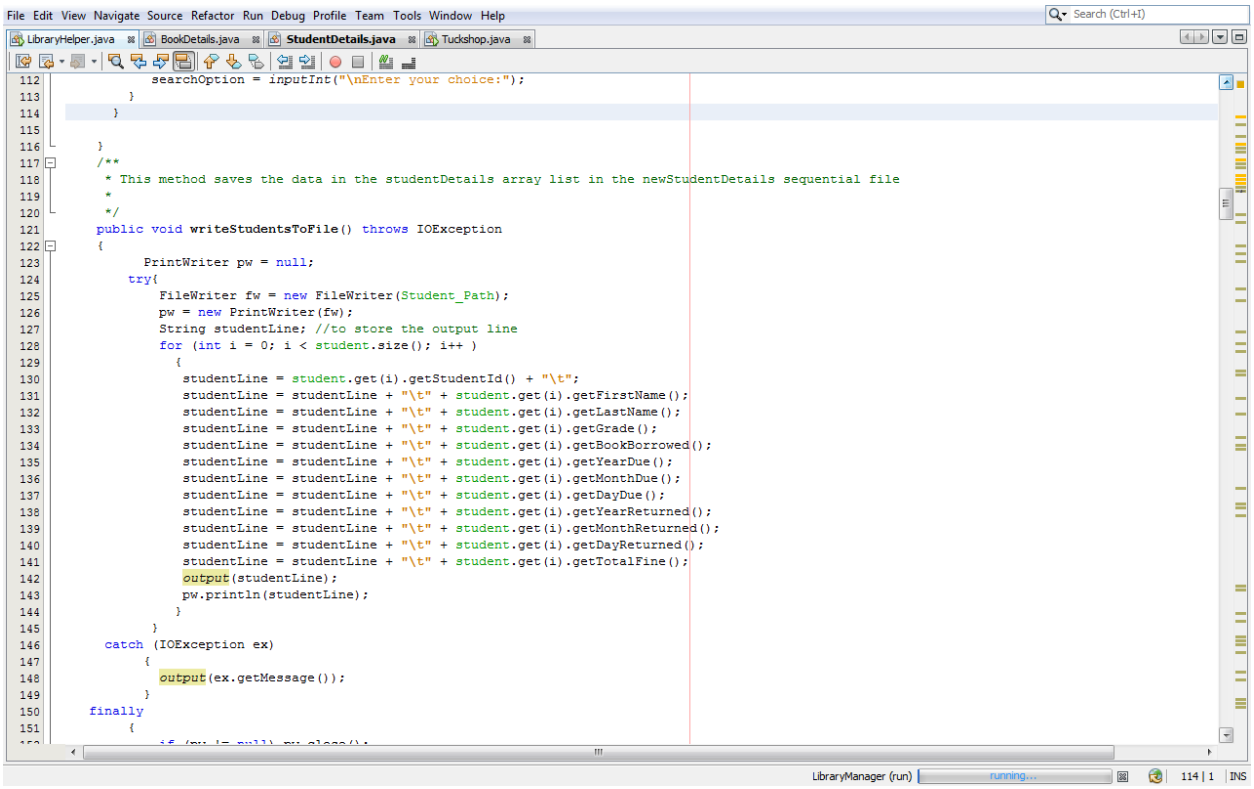
```
39
40          //validation check toensure that the inputted username matches the valid username required for the program to run
41          int usernameValidation; //this is the variable that will compare the inputted username to the correct username
42          do
43          {
44          inputUsername = inputString (" Please enter the correct username:");
45          usernameValidation = correctUsername.compareTo(inputUsername); //compares the inputted username to the correct username
46          } while (usernameValidation != 0);
47          output("Username confirmed");
48
49          //validation check to ensure that the inputted password matches the valid password required for the program to run
50          int passwordValidation;
51          do
52          {
53           inputPassword = inputString (" Please enter the correct password:");
54           passwordValidation = correctPassword.compareTo(inputPassword); //comparres the inputted password to the correct password
55          } while (passwordValidation != 0);
56          output("Password confirmed");
57
58          loadBooks(); // method to load the newBookDetails sequential file
59          loadStudents(); //method to load the newBookDetails sequential file
60
61          // display the main menu
62          output ("\n Welcome to the Library Manager Main Menu" );
63          output("1. Borrow a book");
64          output("2. Return a book");
65          output("3. Re-stock the library");
66          output("4. Search");
67
68          int option = inputInt("Enter the number of the option you want to undertake:");
69
70          if (option == 1)
71          borrowBook(); //method that is run when the librarian chooses the option to borrow a book
72
73          if(option == 2)
74          returnBook(); //method that is run when the librarian chooses the optoin to return a book
75
76          if (option == 3)
77          AddBook();  //method that is run when the librarian chooses the option to re-stock the library
78
```

4.

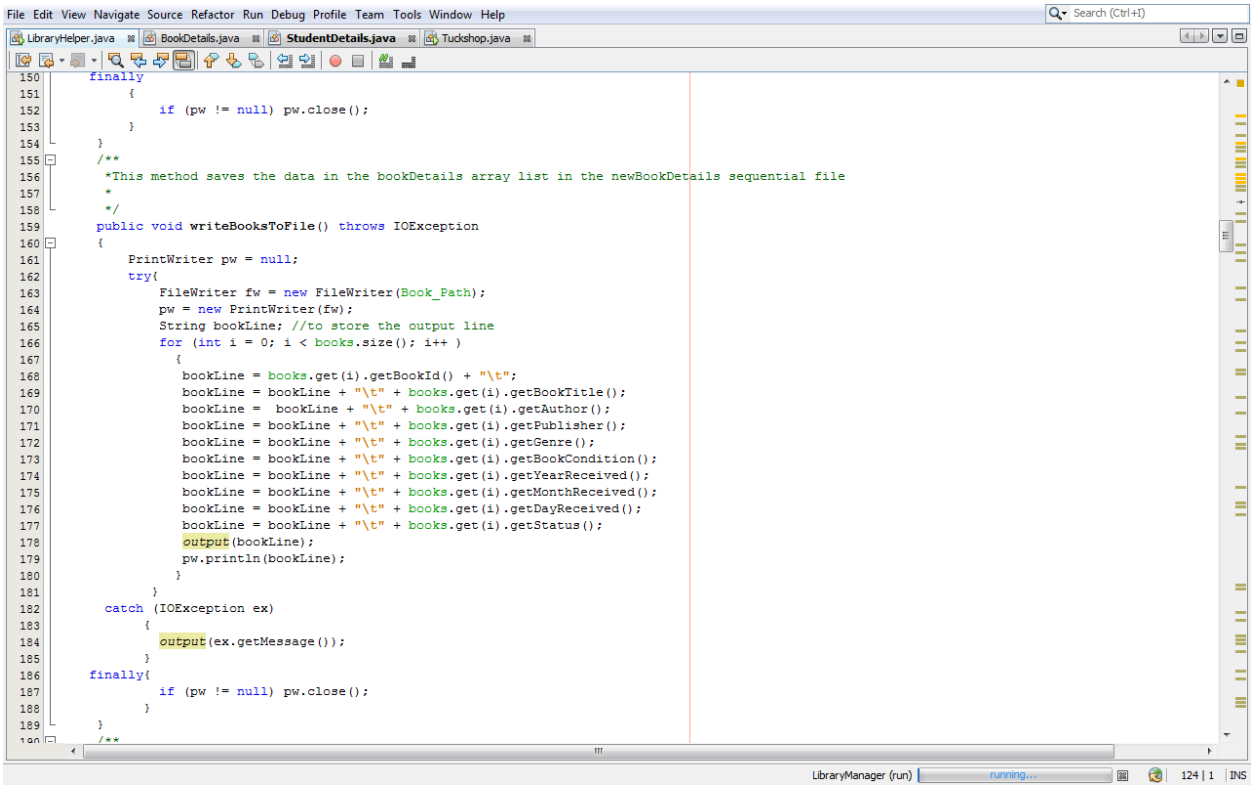


```
75
76          if (option == 3)
77          AddBook();  //method that is run when the librarian chooses the option to re-stock the library
78
79         if(option == 4)
80         {
81          //Display the search menu
82          output("\n Welcome to the Searching Menu");
83          output("1.Search book by author");
84          output("2.Search book by publisher");
85          output("3. Search book by genre");
86          output("4. Search all students who have been fined");
87          int searchOption = inputInt("Enter the number of the option you want to undertake");
88          while(searchOption > 0)
89          {
90           try
91            {
92             if(searchOption == 1)
93                 searchBookByAuthor(); //method that is run when the librarian chooses the option to search for books by their authors
94
95             if(searchOption == 2)
96                 searchBookByPublisher(); //method that is run when the librarian chooses the option to search for books by their publishers
97
98             if(searchOption == 3)
99                 searchBookByGenre();//method that is run when the librarian chooses the option to search for books by their genre
100
101            if(searchOption == 4)
102                searchStudentByFine();//method that is run when the librarian chooses the option to search for fined students
103           }
104           catch(Exception e)
105           {
106            output(e.getMessage());
107           }
108           //Display the menu
109             output("\nNEW Searching Menu");
110             output("Enter the number of the new search option you want to undertake");
111             output("Or press '0' to return to the main menu");
112            searchOption = inputInt("\nEnter your choice:");
113         }
114        }
```

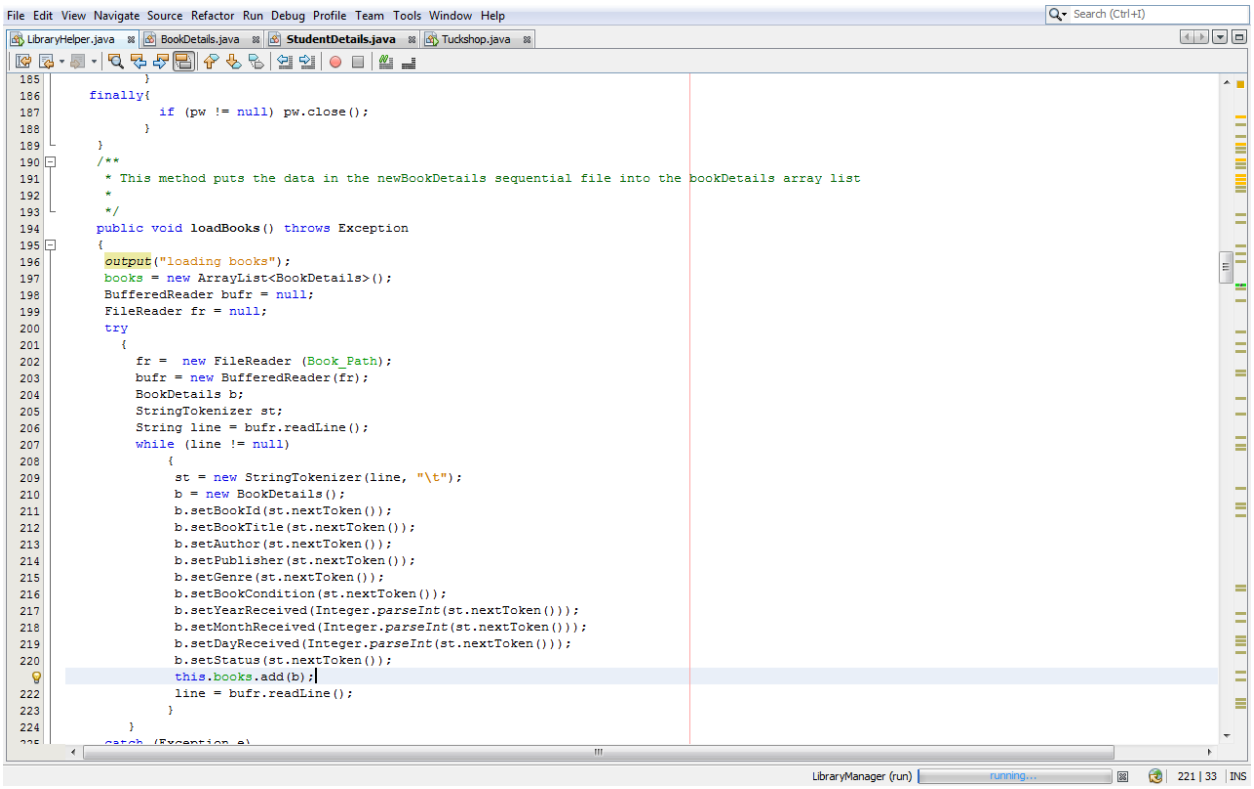
5.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
searchOption = inputInt("\nEnter your choice:");
}
}
}
/**
* This method saves the data in the studentDetails array list in the newStudentDetails sequential file
*
*/
public void writeStudentsToFile() throws IOException
{
PrintWriter pw = null;
try{
FileWriter fw = new FileWriter(Student_Path);
pw = new PrintWriter(fw);
String studentLine; //to store the output line
for (int i = 0; i < student.size(); i++)
{
studentLine = student.get(i).getStudentId() + "\t";
studentLine = studentLine + "\t" + student.get(i).getFirstName();
studentLine = studentLine + "\t" + student.get(i).getLastName();
studentLine = studentLine + "\t" + student.get(i).getGrade();
studentLine = studentLine + "\t" + student.get(i).getBookBorrowed();
studentLine = studentLine + "\t" + student.get(i).getYearDue();
studentLine = studentLine + "\t" + student.get(i).getMonthDue();
studentLine = studentLine + "\t" + student.get(i).getDayDue();
studentLine = studentLine + "\t" + student.get(i).getYearReturned();
studentLine = studentLine + "\t" + student.get(i).getMonthReturned();
studentLine = studentLine + "\t" + student.get(i).getDayReturned();
studentLine = studentLine + "\t" + student.get(i).getTotalFine();
output(studentLine);
pw.println(studentLine);
}
}
catch (IOException ex)
{
output(ex.getMessage());
}
finally
{
LibraryManager (run)
running...
114 | 1
INS

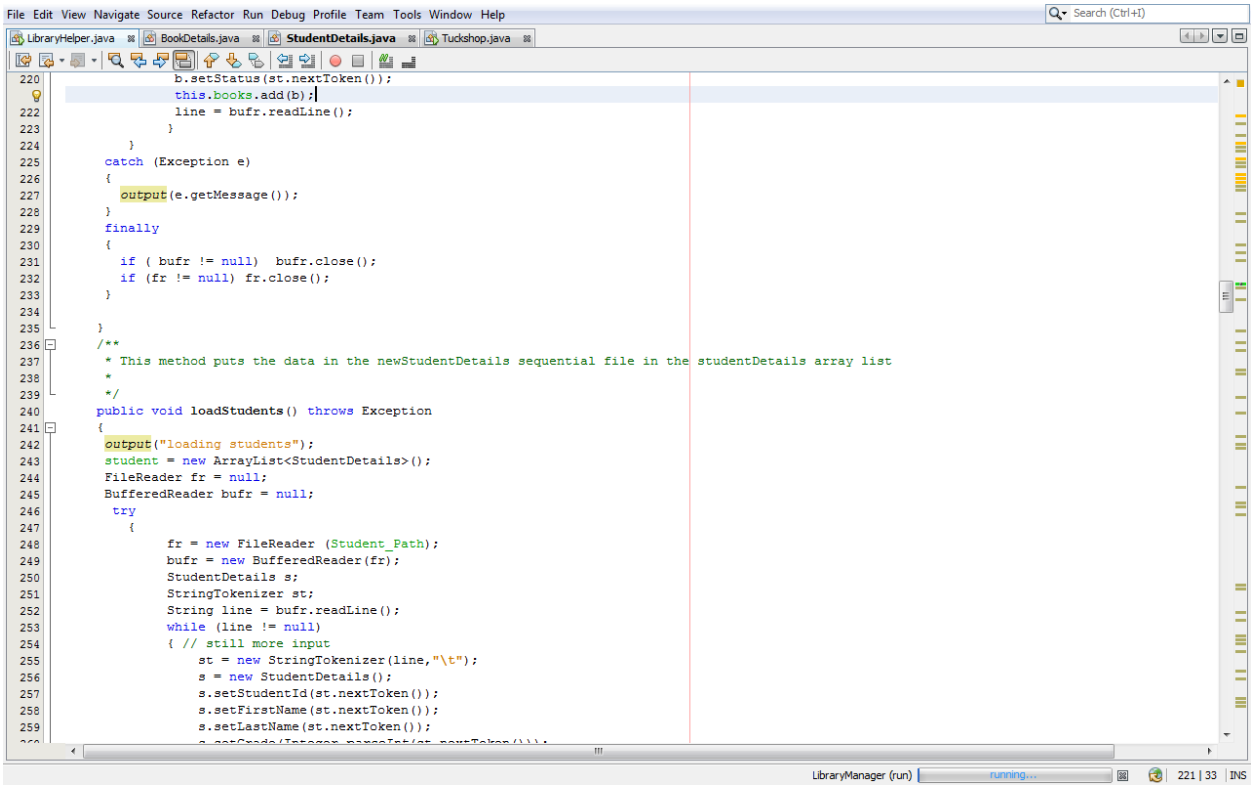
6.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
finally
{
if (pw != null) pw.close();
}
}
/**
*This method saves the data in the bookDetails array list in the newBookDetails sequential file
*
*/
public void writeBooksToFile() throws IOException
{
PrintWriter pw = null;
try{
FileWriter fw = new FileWriter(Book_Path);
pw = new PrintWriter(fw);
String bookLine; //to store the output line
for (int i = 0; i < books.size(); i++)
{
bookLine = books.get(i).getBookId() + "\t";
bookLine = bookLine + "\t" + books.get(i).getBookTitle();
bookLine = bookLine + "\t" + books.get(i).getAuthor();
bookLine = bookLine + "\t" + books.get(i).getPublisher();
bookLine = bookLine + "\t" + books.get(i).getGenre();
bookLine = bookLine + "\t" + books.get(i).getBookCondition();
bookLine = bookLine + "\t" + books.get(i).getYearReceived();
bookLine = bookLine + "\t" + books.get(i).getMonthReceived();
bookLine = bookLine + "\t" + books.get(i).getDayReceived();
bookLine = bookLine + "\t" + books.get(i).getStatus();
output(bookLine);
pw.println(bookLine);
}
}
catch (IOException ex)
{
output(ex.getMessage());
}
finally{
if (pw != null) pw.close();
}
}
/**
LibraryManager (run)
running...
124 | 1
INS

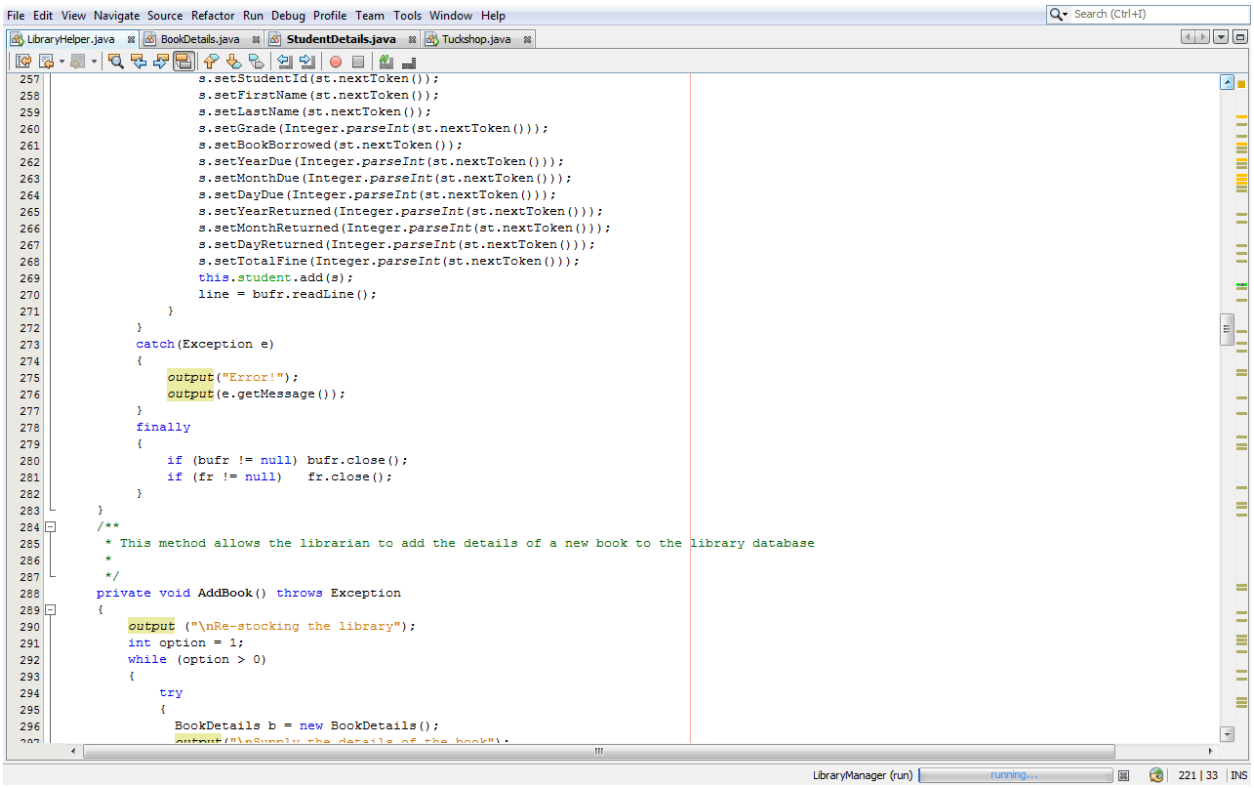
7.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
}
finally{
if (pw != null) pw.close();
}
}
/**
* This method puts the data in the newBookDetails sequential file into the bookDetails array list
*
*/
public void loadBooks() throws Exception
{
output("loading books");
books = new ArrayList<BookDetails>();
BufferedReader bufr = null;
FileReader fr = null;
try
{
fr = new FileReader (Book_Path);
bufr = new BufferedReader(fr);
BookDetails b;
StringTokenizer st;
String line = bufr.readLine();
while (line != null)
{
st = new StringTokenizer(line, "\t");
b = new BookDetails();
b.setBookId(st.nextToken());
b.setBookTitle(st.nextToken());
b.setAuthor(st.nextToken());
b.setPublisher(st.nextToken());
b.setGenre(st.nextToken());
b.setBookCondition(st.nextToken());
b.setYearReceived(Integer.parseInt(st.nextToken()));
b.setMonthReceived(Integer.parseInt(st.nextToken()));
b.setDayReceived(Integer.parseInt(st.nextToken()));
b.setStatus(st.nextToken());
this.books.add(b);
line = bufr.readLine();
}
}
LibraryManager (run)
running...
221 | 33
INS

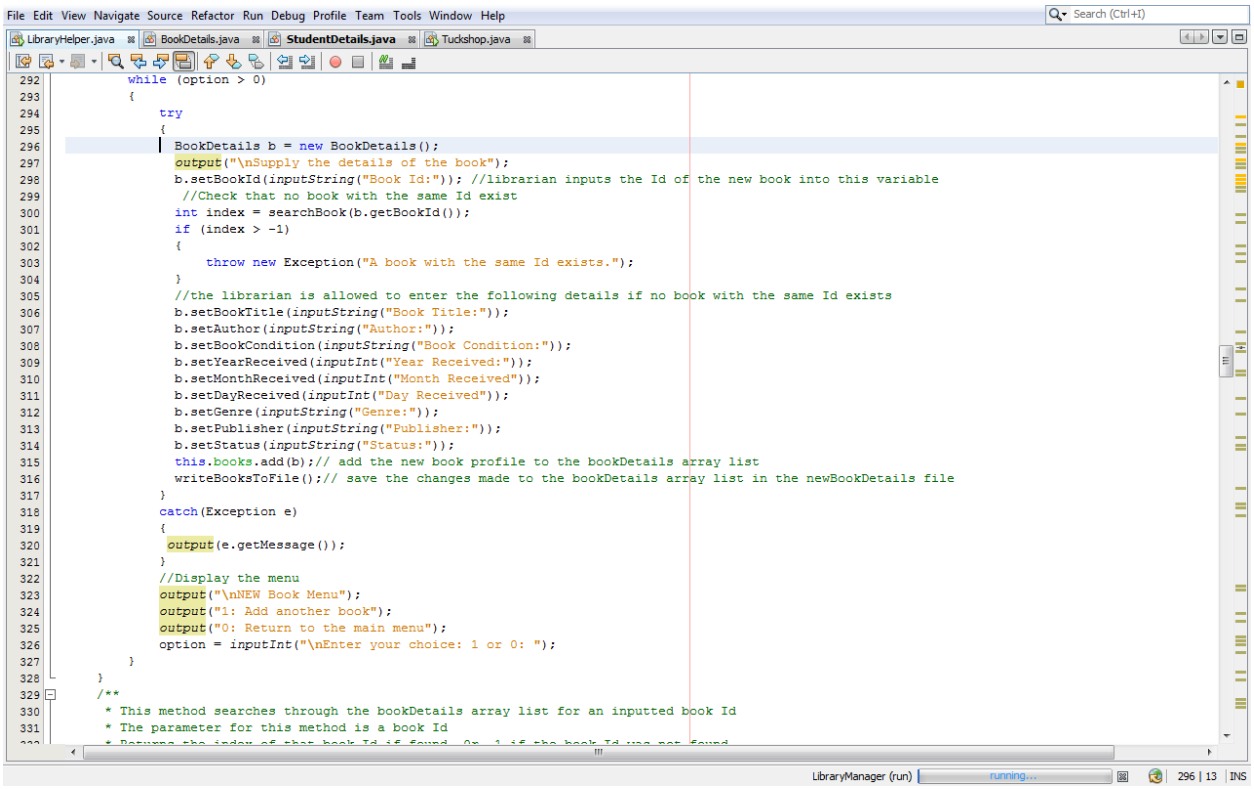
8.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
b.setStatus(st.nextToken());
this.books.add(b);
line = bufr.readLine();
}
}
catch (Exception e)
{
output(e.getMessage());
}
finally
{
if (bufr != null) bufr.close();
if (fr != null) fr.close();
}
}
/**
* This method puts the data in the newStudentDetails sequential file in the studentDetails array list
*
*/
public void loadStudents() throws Exception
{
output("loading students");
student = new ArrayList<StudentDetails>();
FileReader fr = null;
BufferedReader bufr = null;
try
{
fr = new FileReader (Student_Path);
bufr = new BufferedReader(fr);
StudentDetails s;
StringTokenizer st;
String line = bufr.readLine();
while (line != null)
{ // still more input
st = new StringTokenizer(line,"\t");
s = new StudentDetails();
s.setStudentId(st.nextToken());
s.setFirstName(st.nextToken());
s.setLastName(st.nextToken());
LibraryManager (run)
running...
221 | 33
INS

9.

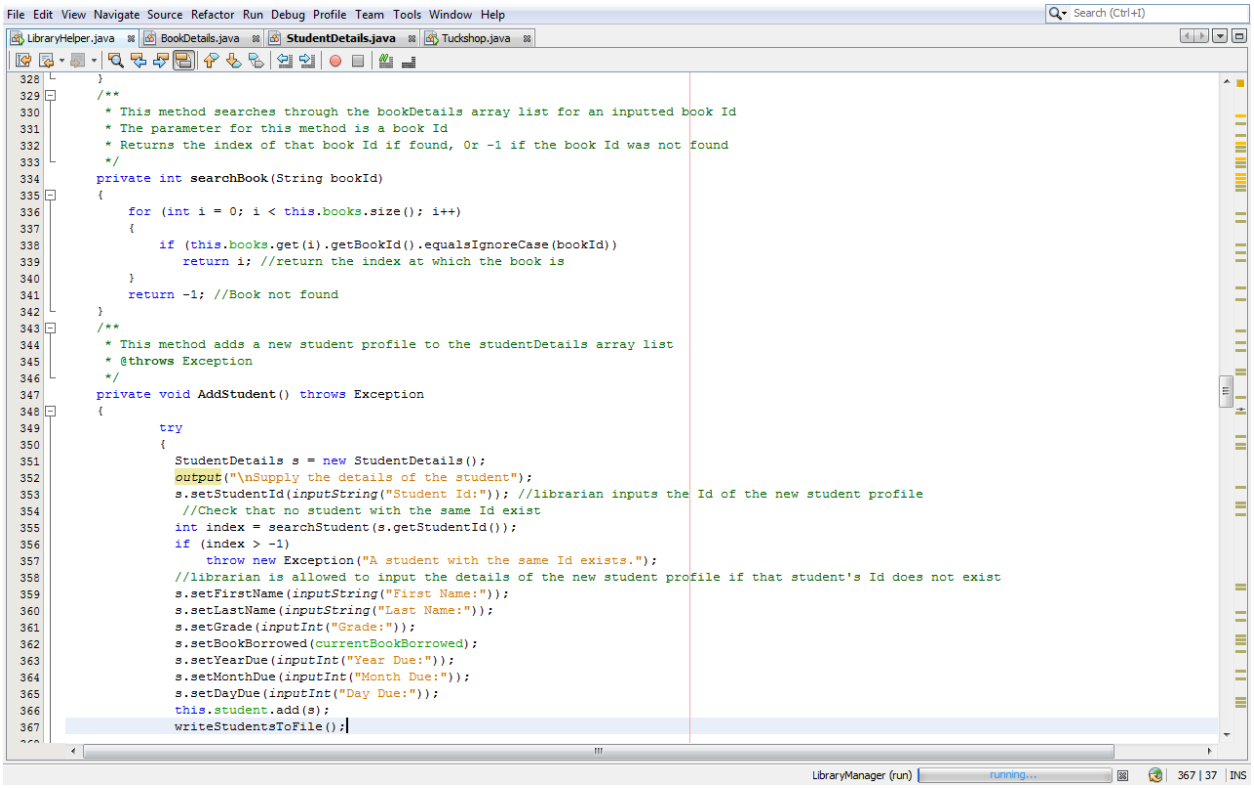

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
s.setStudentId(st.nextToken());
s.setFirstName(st.nextToken());
s.setLastName(st.nextToken());
s.setGrade(Integer.parseInt(st.nextToken()));
s.setBookBorrowed(st.nextToken());
s.setYearDue(Integer.parseInt(st.nextToken()));
s.setMonthDue(Integer.parseInt(st.nextToken()));
s.setDayDue(Integer.parseInt(st.nextToken()));
s.setYearReturned(Integer.parseInt(st.nextToken()));
s.setMonthReturned(Integer.parseInt(st.nextToken()));
s.setDayReturned(Integer.parseInt(st.nextToken()));
s.setTotalFine(Integer.parseInt(st.nextToken()));
this.student.add(s);
line = bufr.readLine();
}
}
catch(Exception e)
{
output("Error!");
output(e.getMessage());
}
finally
{
if (bufr != null) bufr.close();
if (fr != null) fr.close();
}
}
/**
* This method allows the librarian to add the details of a new book to the library database
*
*/
private void AddBook() throws Exception
{
output ("\nRe-stocking the library");
int option = 1;
while (option > 0)
{
try
{
BookDetails b = new BookDetails();
LibraryManager (run)
running...
221 | 33
INS

10.



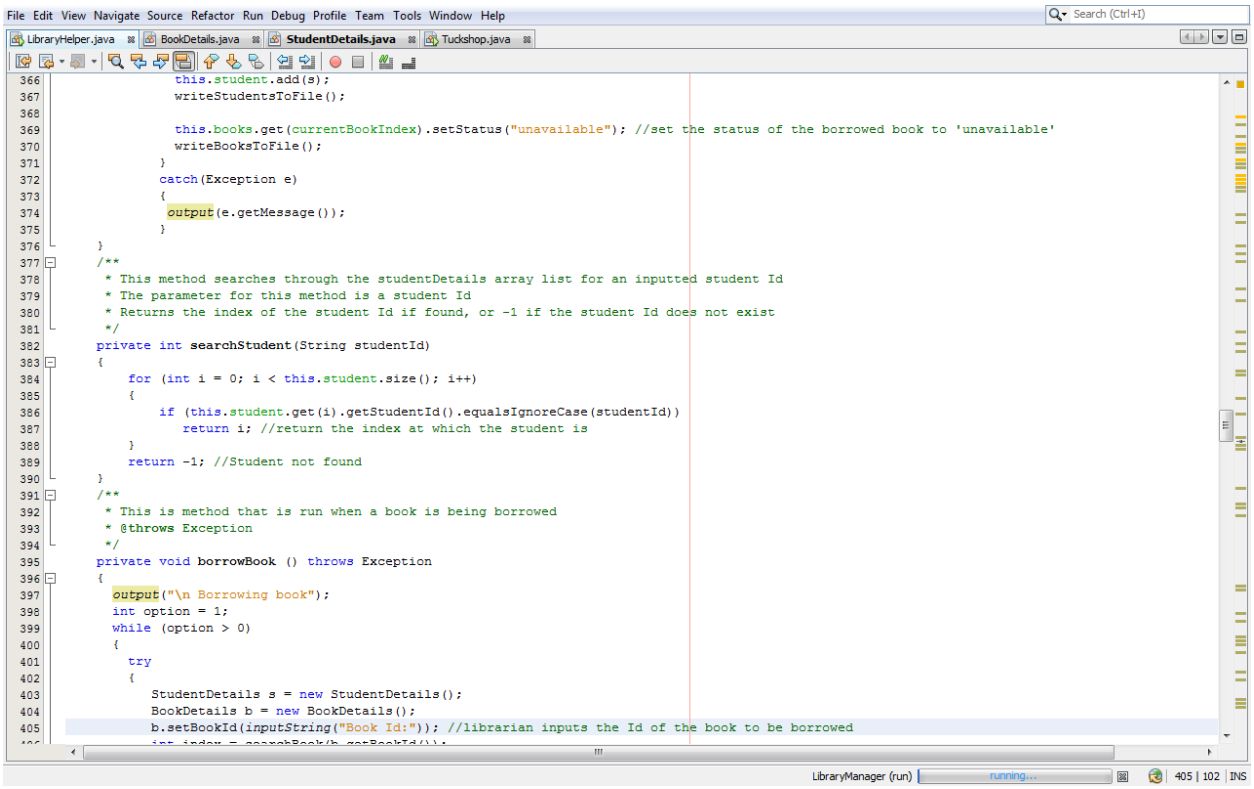

```
292         while (option > 0)
293         {
294             try
295             {
296               BookDetails b = new BookDetails();
297               output("\nSupply the details of the book");
298               b.setBookId(inputString("Book Id:")); //librarian inputs the Id of the new book into this variable
299                //Check that no book with the same Id exist
300               int index = searchBook(b.getBookId());
301               if (index > -1)
302               {
303                   throw new Exception("A book with the same Id exists.");
304               }
305               //the librarian is allowed to enter the following details if no book with the same Id exists
306               b.setBookTitle(inputString("Book Title:"));
307               b.setAuthor(inputString("Author:"));
308               b.setBookCondition(inputString("Book Condition:"));
309               b.setYearReceived(inputInt("Year Received:"));
310               b.setMonthReceived(inputInt("Month Received"));
311               b.setDayReceived(inputInt("Day Received"));
312               b.setGenre(inputString("Genre:"));
313               b.setPublisher(inputString("Publisher:"));
314               b.setStatus(inputString("Status:"));
315               this.books.add(b);// add the new book profile to the bookDetails array list
316               writeBooksToFile();// save the changes made to the bookDetails array list in the newBookDetails file
317             }
318             catch(Exception e)
319             {
320              output(e.getMessage());
321             }
322             //Display the menu
323             output("\nNEW Book Menu");
324             output("1: Add another book");
325             output("0: Return to the main menu");
326             option = inputInt("\nEnter your choice: 1 or 0: ");
327         }
328     }
329     /**
330      * This method searches through the bookDetails array list for an inputted book Id
331      * The parameter for this method is a book Id
```

11.



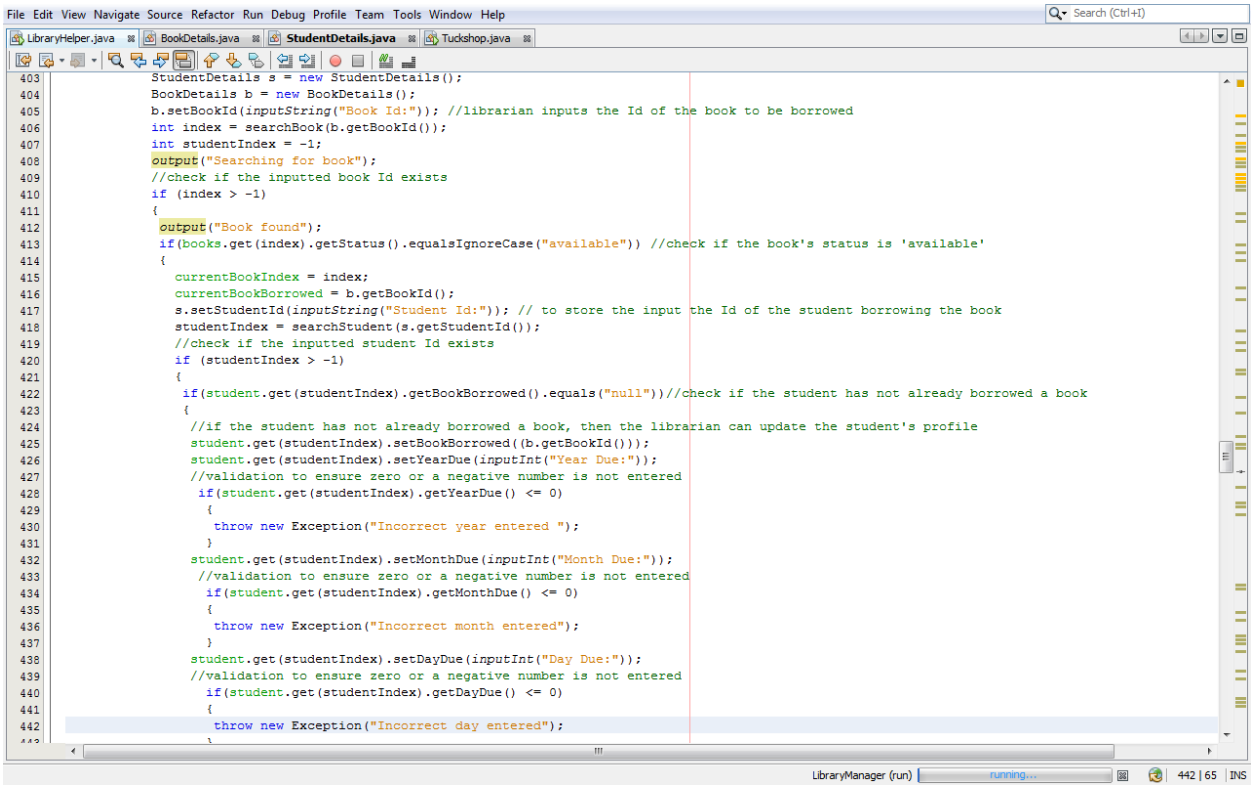

```
328     }
329     /**
330      * This method searches through the bookDetails array list for an inputted book Id
331      * The parameter for this method is a book Id
332      * Returns the index of that book Id if found, 0r -1 if the book Id was not found
333      */
334     private int searchBook(String bookId)
335     {
336         for (int i = 0; i < this.books.size(); i++)
337         {
338             if (this.books.get(i).getBookId().equalsIgnoreCase(bookId))
339                 return i; //return the index at which the book is
340         }
341         return -1; //Book not found
342     }
343     /**
344      * This method adds a new student profile to the studentDetails array list
345      * @throws Exception
346      */
347     private void AddStudent() throws Exception
348     {
349             try
350             {
351               StudentDetails s = new StudentDetails();
352               output("\nSupply the details of the student");
353               s.setStudentId(inputString("Student Id:")); //librarian inputs the Id of the new student profile
354                //Check that no student with the same Id exist
355               int index = searchStudent(s.getStudentId());
356               if (index > -1)
357                   throw new Exception("A student with the same Id exists.");
358               //librarian is allowed to input the details of the new student profile if that student's Id does not exist
359               s.setFirstName(inputString("First Name:"));
360               s.setLastName(inputString("Last Name:"));
361               s.setGrade(inputInt("Grade:"));
362               s.setBookBorrowed(currentBookBorrowed);
363               s.setYearDue(inputInt("Year Due:"));
364               s.setMonthDue(inputInt("Month Due:"));
365               s.setDayDue(inputInt("Day Due:"));
366               this.student.add(s);
367               writeStudentsToFile();
```

12.



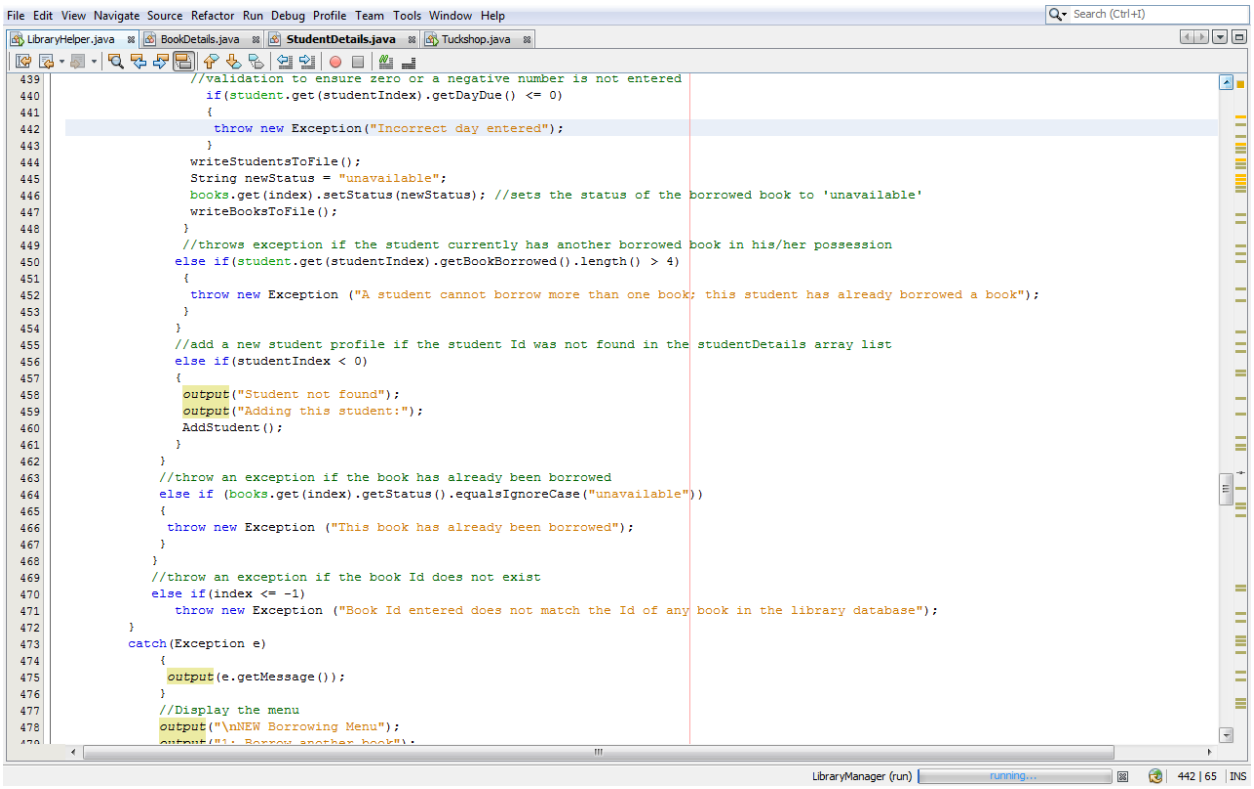
```
                this.student.add(s);
                writeStudentsToFile();

                this.books.get(currentBookIndex).setStatus("unavailable"); //set the status of the borrowed book to 'unavailable'
                writeBooksToFile();
              }
              catch(Exception e)
              {
               output(e.getMessage());
              }
    }
    /**
     * This method searches through the studentDetails array list for an inputted student Id
     * The parameter for this method is a student Id
     * Returns the index of the student Id if found, or -1 if the student Id does not exist
     */
    private int searchStudent(String studentId)
    {
        for (int i = 0; i < this.student.size(); i++)
        {
            if (this.student.get(i).getStudentId().equalsIgnoreCase(studentId))
               return i; //return the index at which the student is
        }
        return -1; //Student not found
    }
    /**
     * This is method that is run when a book is being borrowed
     * @throws Exception
     */
    private void borrowBook () throws Exception
    {
      output("\n Borrowing book");
      int option = 1;
      while (option > 0)
      {
        try
        {
           StudentDetails s = new StudentDetails();
           BookDetails b = new BookDetails();
           b.setBookId(inputString("Book Id:")); //librarian inputs the Id of the book to be borrowed
```


13.



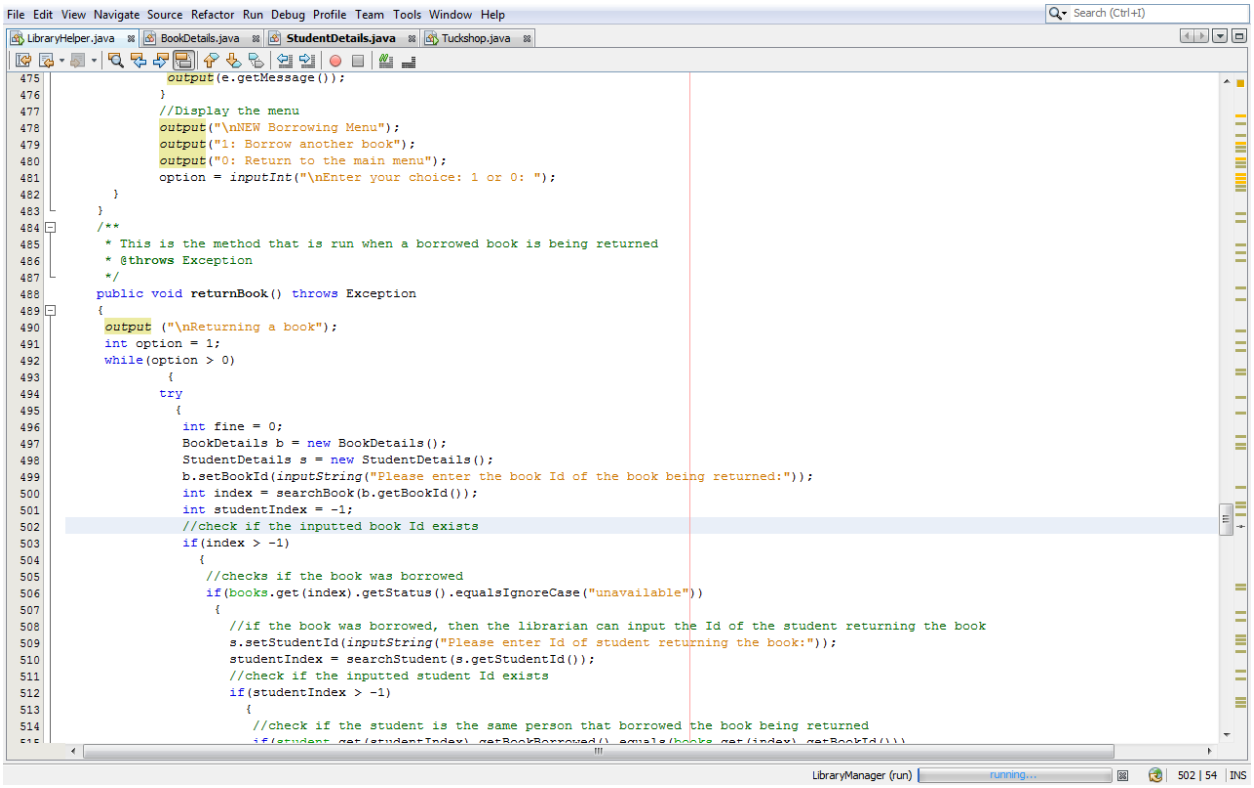
```
           StudentDetails s = new StudentDetails();
           BookDetails b = new BookDetails();
           b.setBookId(inputString("Book Id:")); //librarian inputs the Id of the book to be borrowed
           int index = searchBook(b.getBookId());
           int studentIndex = -1;
           output("Searching for book");
           //check if the inputted book Id exists
           if (index > -1)
           {
            output("Book found");
            if(books.get(index).getStatus().equalsIgnoreCase("available")) //check if the book's status is 'available'
            {
              currentBookIndex = index;
              currentBookBorrowed = b.getBookId();
              s.setStudentId(inputString("Student Id:")); // to store the input the Id of the student borrowing the book
              studentIndex = searchStudent(s.getStudentId());
              //check if the inputted student Id exists
              if (studentIndex > -1)
              {
               if(student.get(studentIndex).getBookBorrowed().equals("null"))//check if the student has not already borrowed a book
               {
                //if the student has not already borrowed a book, then the librarian can update the student's profile
                student.get(studentIndex).setBookBorrowed((b.getBookId()));
                student.get(studentIndex).setYearDue(inputInt("Year Due:"));
                //validation to ensure zero or a negative number is not entered
                 if(student.get(studentIndex).getYearDue() <= 0)
                  {
                   throw new Exception("Incorrect year entered ");
                  }
                student.get(studentIndex).setMonthDue(inputInt("Month Due:"));
                 //validation to ensure zero or a negative number is not entered
                  if(student.get(studentIndex).getMonthDue() <= 0)
                  {
                   throw new Exception("Incorrect month entered");
                  }
                student.get(studentIndex).setDayDue(inputInt("Day Due:"));
                //validation to ensure zero or a negative number is not entered
                  if(student.get(studentIndex).getDayDue() <= 0)
                  {
                   throw new Exception("Incorrect day entered");
```


14.



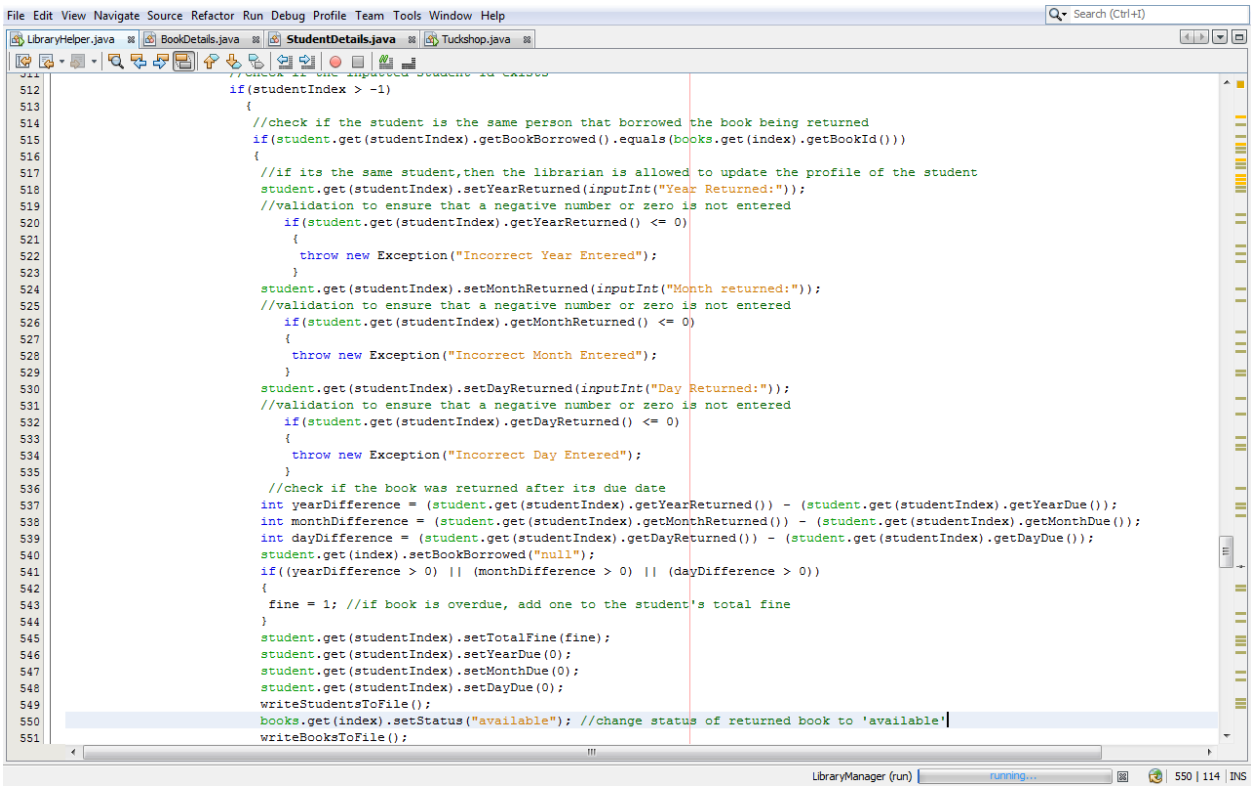
```
                //validation to ensure zero or a negative number is not entered
                  if(student.get(studentIndex).getDayDue() <= 0)
                  {
                   throw new Exception("Incorrect day entered");
                  }
                 writeStudentsToFile();
                 String newStatus = "unavailable";
                 books.get(index).setStatus(newStatus); //sets the status of the borrowed book to 'unavailable'
                 writeBooksToFile();
                }
                //throws exception if the student currently has another borrowed book in his/her possession
               else if(student.get(studentIndex).getBookBorrowed().length() > 4)
                {
                 throw new Exception ("A student cannot borrow more than one book; this student has already borrowed a book");
                }
               }
               //add a new student profile if the student Id was not found in the studentDetails array list
               else if(studentIndex < 0)
               {
                output("Student not found");
                output("Adding this student:");
                AddStudent();
               }
              }
              //throw an exception if the book has already been borrowed
              else if (books.get(index).getStatus().equalsIgnoreCase("unavailable"))
              {
               throw new Exception ("This book has already been borrowed");
              }
             }
             //throw an exception if the book Id does not exist
             else if(index <= -1)
                throw new Exception ("Book Id entered does not match the Id of any book in the library database");
          }
          catch(Exception e)
              {
               output(e.getMessage());
              }
              //Display the menu
              output("\nNEW Borrowing Menu");
```

15.



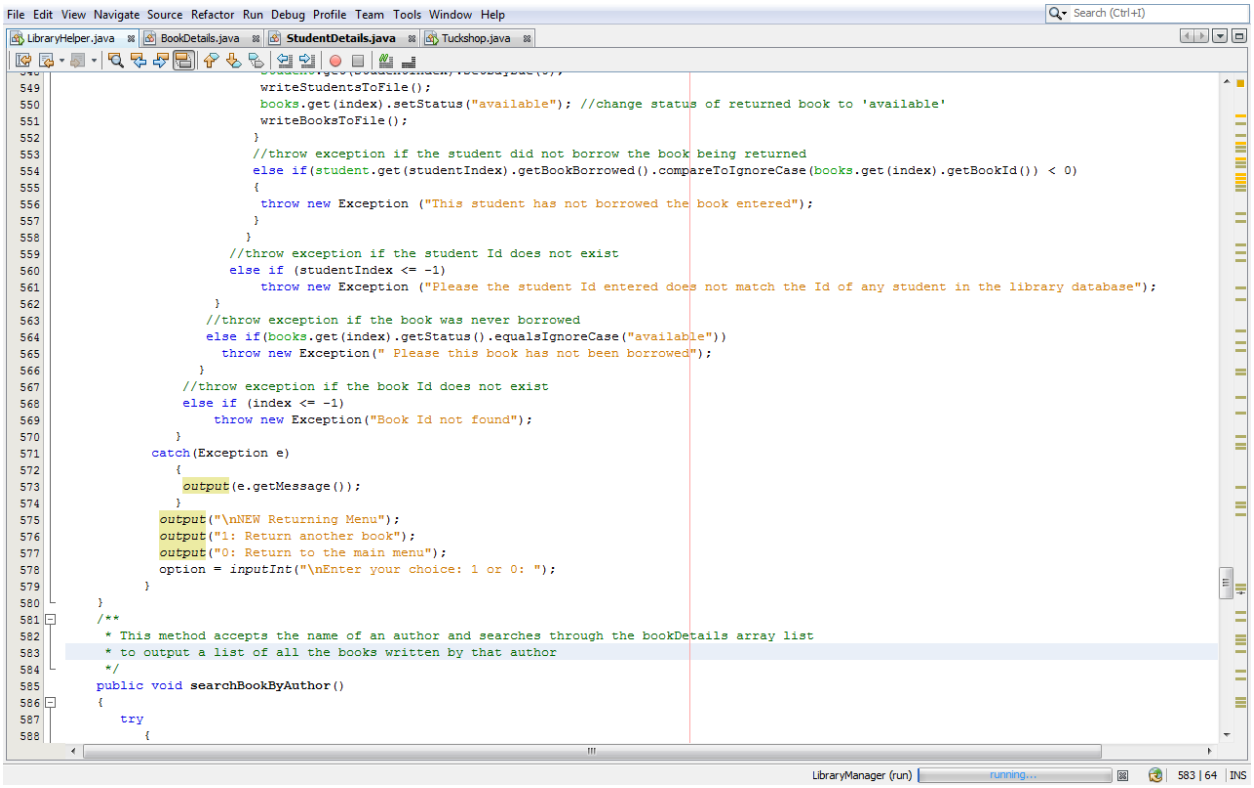
```
              output(e.getMessage());
             }
             //Display the menu
             output("\nNEW Borrowing Menu");
             output("1: Borrow another book");
             output("0: Return to the main menu");
             option = inputInt("\nEnter your choice: 1 or 0: ");
        }
    }
    /**
     * This is the method that is run when a borrowed book is being returned
     * @throws Exception
     */
    public void returnBook() throws Exception
    {
     output ("\nReturning a book");
     int option = 1;
     while(option > 0)
             {
            try
              {
               int fine = 0;
               BookDetails b = new BookDetails();
               StudentDetails s = new StudentDetails();
               b.setBookId(inputString("Please enter the book Id of the book being returned:"));
               int index = searchBook(b.getBookId());
               int studentIndex = -1;
               //check if the inputted book Id exists
               if(index > -1)
                 {
                  //checks if the book was borrowed
                  if(books.get(index).getStatus().equalsIgnoreCase("unavailable"))
                   {
                     //if the book was borrowed, then the librarian can input the Id of the student returning the book
                     s.setStudentId(inputString("Please enter Id of student returning the book:"));
                     studentIndex = searchStudent(s.getStudentId());
                     //check if the inputted student Id exists
                     if(studentIndex > -1)
                       {
                        //check if the student is the same person that borrowed the book being returned
```

16.



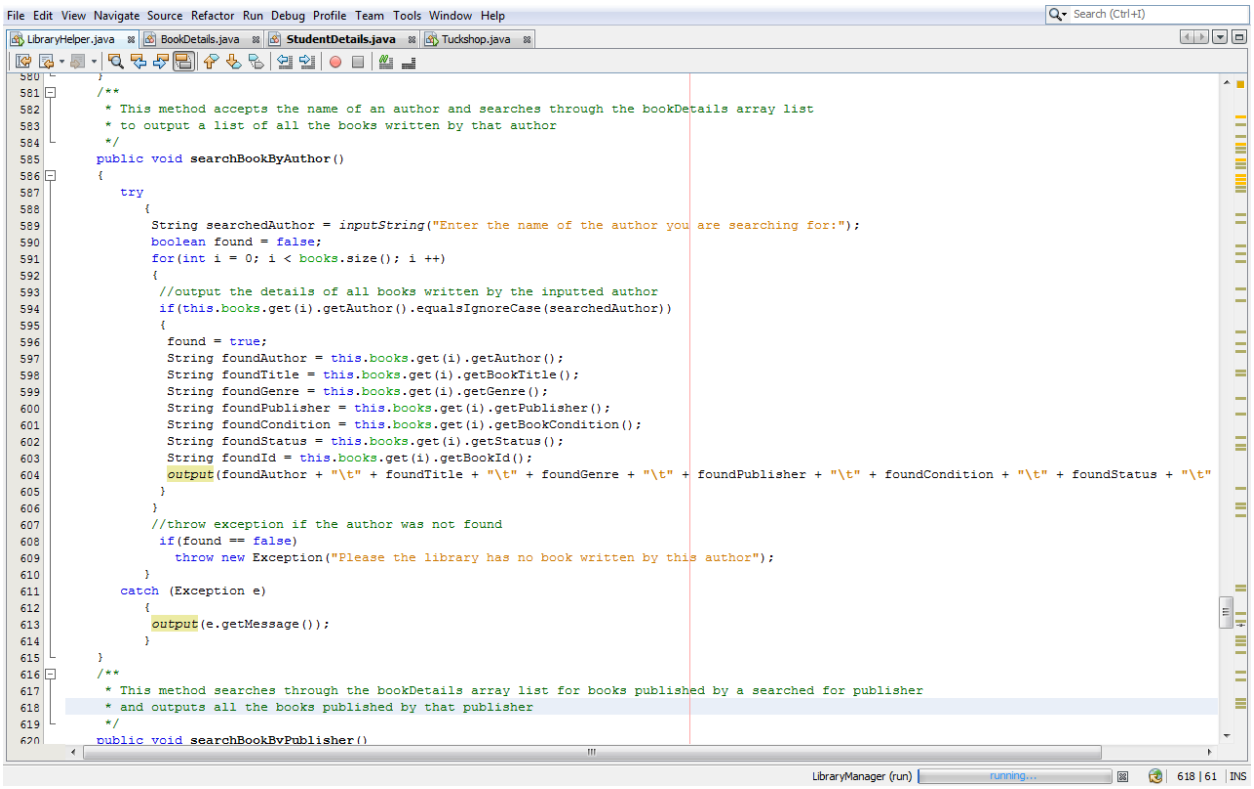
```
                    if(studentIndex > -1)
                      {
                       //check if the student is the same person that borrowed the book being returned
                       if(student.get(studentIndex).getBookBorrowed().equals(books.get(index).getBookId()))
                       {
                        //if its the same student,then the librarian is allowed to update the profile of the student
                        student.get(studentIndex).setYearReturned(inputInt("Year Returned:"));
                        //validation to ensure that a negative number or zero is not entered
                           if(student.get(studentIndex).getYearReturned() <= 0)
                            {
                             throw new Exception("Incorrect Year Entered");
                            }
                        student.get(studentIndex).setMonthReturned(inputInt("Month returned:"));
                        //validation to ensure that a negative number or zero is not entered
                           if(student.get(studentIndex).getMonthReturned() <= 0)
                           {
                            throw new Exception("Incorrect Month Entered");
                           }
                        student.get(studentIndex).setDayReturned(inputInt("Day Returned:"));
                        //validation to ensure that a negative number or zero is not entered
                           if(student.get(studentIndex).getDayReturned() <= 0)
                           {
                            throw new Exception("Incorrect Day Entered");
                           }
                         //check if the book was returned after its due date
                        int yearDifference = (student.get(studentIndex).getYearReturned()) - (student.get(studentIndex).getYearDue());
                        int monthDifference = (student.get(studentIndex).getMonthReturned()) - (student.get(studentIndex).getMonthDue());
                        int dayDifference = (student.get(studentIndex).getDayReturned()) - (student.get(studentIndex).getDayDue());
                        student.get(index).setBookBorrowed("null");
                        if((yearDifference > 0) || (monthDifference > 0) || (dayDifference > 0))
                        {
                         fine = 1; //if book is overdue, add one to the student's total fine
                        }
                        student.get(studentIndex).setTotalFine(fine);
                        student.get(studentIndex).setYearDue(0);
                        student.get(studentIndex).setMonthDue(0);
                        student.get(studentIndex).setDayDue(0);
                        writeStudentsToFile();
                        books.get(index).setStatus("available"); //change status of returned book to 'available'
                        writeBooksToFile();
```

17.



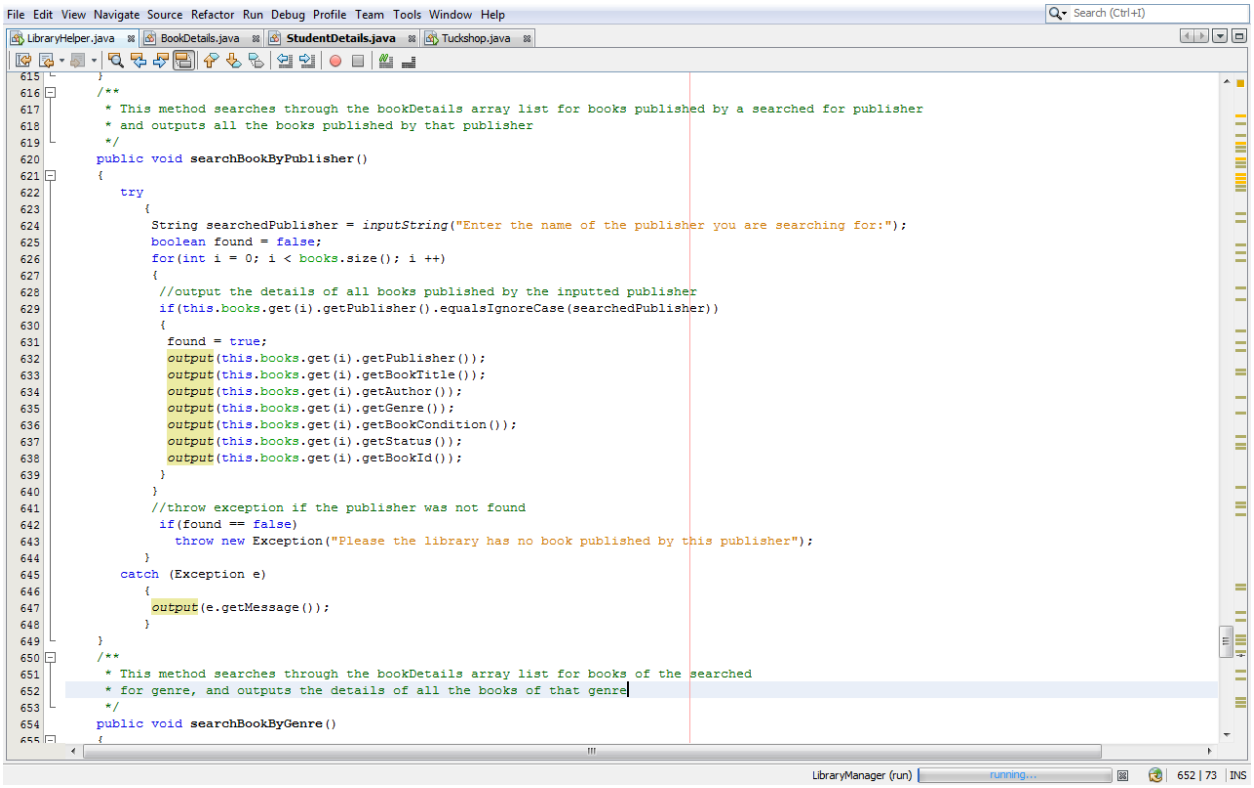
```
                        writeStudentsToFile();
                        books.get(index).setStatus("available"); //change status of returned book to 'available'
                        writeBooksToFile();
                       }
                       //throw exception if the student did not borrow the book being returned
                       else if(student.get(studentIndex).getBookBorrowed().compareToIgnoreCase(books.get(index).getBookId()) < 0)
                       {
                        throw new Exception ("This student has not borrowed the book entered");
                       }
                      }
                     //throw exception if the student Id does not exist
                     else if (studentIndex <= -1)
                        throw new Exception ("Please the student Id entered does not match the Id of any student in the library database");
                   }
                  //throw exception if the book was never borrowed
                  else if(books.get(index).getStatus().equalsIgnoreCase("available"))
                    throw new Exception(" Please this book has not been borrowed");
                 }
               //throw exception if the book Id does not exist
               else if (index <= -1)
                   throw new Exception("Book Id not found");
              }
          catch(Exception e)
              {
               output(e.getMessage());
              }
           output("\nNEW Returning Menu");
           output("1: Return another book");
           output("0: Return to the main menu");
           option = inputInt("\nEnter your choice: 1 or 0: ");
          }
    }
    /**
     * This method accepts the name of an author and searches through the bookDetails array list
     * to output a list of all the books written by that author
     */
    public void searchBookByAuthor()
    {
       try
          {
```

18.



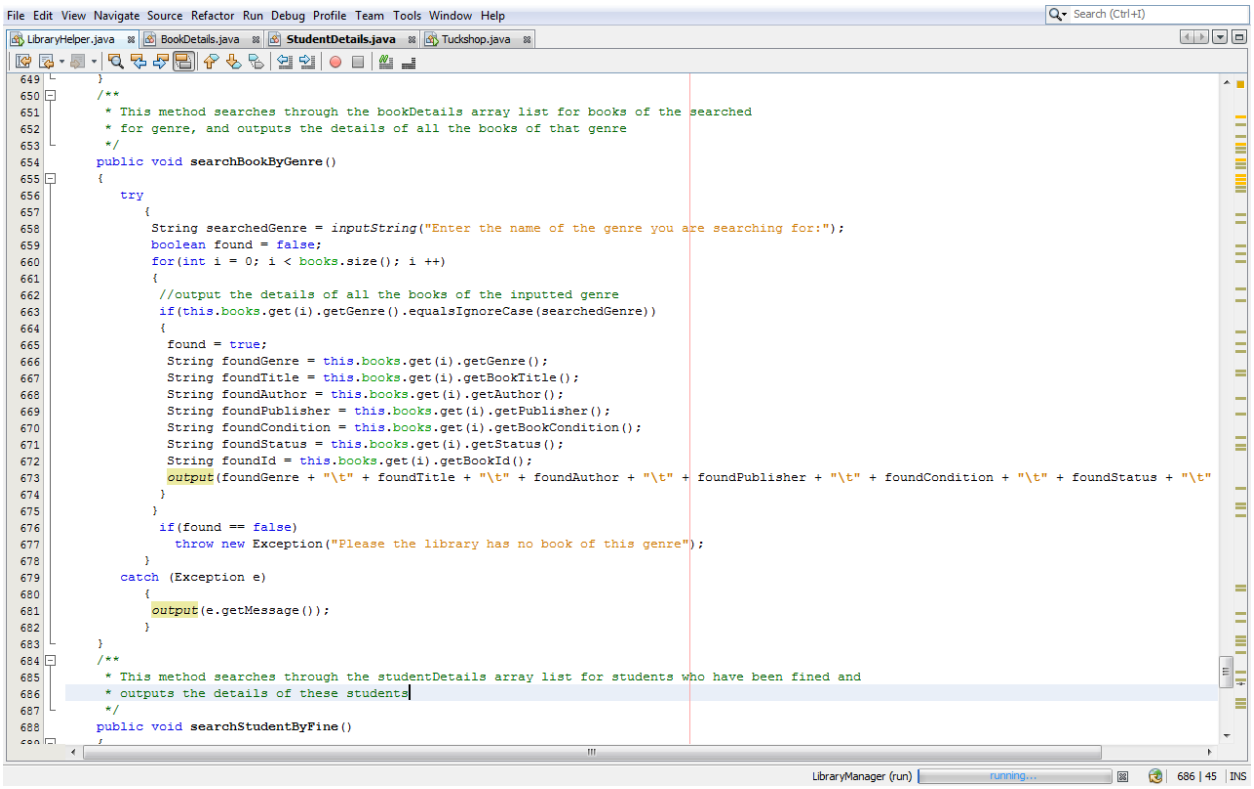

```
    }
    /**
     * This method accepts the name of an author and searches through the bookDetails array list
     * to output a list of all the books written by that author
     */
    public void searchBookByAuthor()
    {
       try
          {
           String searchedAuthor = inputString("Enter the name of the author you are searching for:");
           boolean found = false;
           for(int i = 0; i < books.size(); i ++)
           {
            //output the details of all books written by the inputted author
            if(this.books.get(i).getAuthor().equalsIgnoreCase(searchedAuthor))
            {
             found = true;
             String foundAuthor = this.books.get(i).getAuthor();
             String foundTitle = this.books.get(i).getBookTitle();
             String foundGenre = this.books.get(i).getGenre();
             String foundPublisher = this.books.get(i).getPublisher();
             String foundCondition = this.books.get(i).getBookCondition();
             String foundStatus = this.books.get(i).getStatus();
             String foundId = this.books.get(i).getBookId();
             output(foundAuthor + "\t" + foundTitle + "\t" + foundGenre + "\t" + foundPublisher + "\t" + foundCondition + "\t" + foundStatus + "\t"
            }
           }
           //throw exception if the author was not found
            if(found == false)
              throw new Exception("Please the library has no book written by this author");
          }
       catch (Exception e)
          {
           output(e.getMessage());
          }
    }
    /**
     * This method searches through the bookDetails array list for books published by a searched for publisher
     * and outputs all the books published by that publisher
     */
    public void searchBookByPublisher()
```

19.



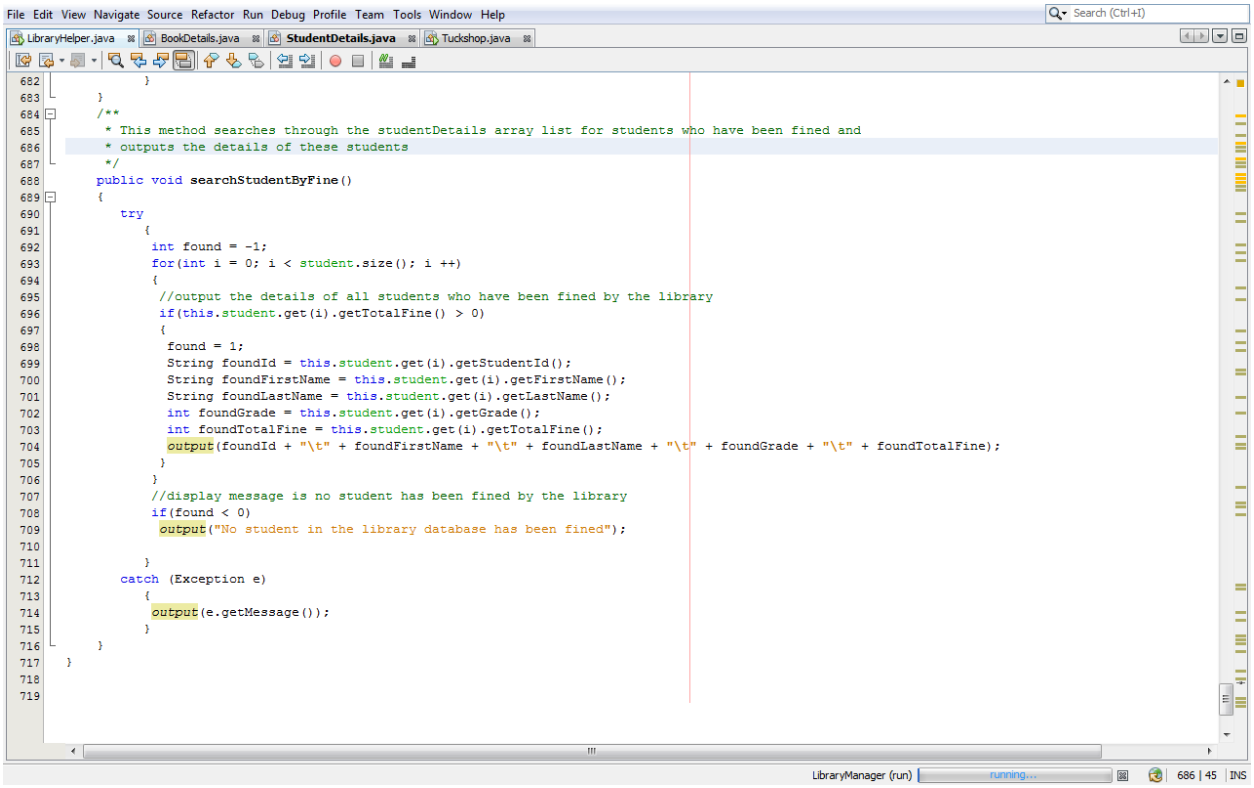

```
    }
    /**
     * This method searches through the bookDetails array list for books published by a searched for publisher
     * and outputs all the books published by that publisher
     */
    public void searchBookByPublisher()
    {
       try
          {
           String searchedPublisher = inputString("Enter the name of the publisher you are searching for:");
           boolean found = false;
           for(int i = 0; i < books.size(); i ++)
           {
            //output the details of all books published by the inputted publisher
            if(this.books.get(i).getPublisher().equalsIgnoreCase(searchedPublisher))
            {
             found = true;
             output(this.books.get(i).getPublisher());
             output(this.books.get(i).getBookTitle());
             output(this.books.get(i).getAuthor());
             output(this.books.get(i).getGenre());
             output(this.books.get(i).getBookCondition());
             output(this.books.get(i).getStatus());
             output(this.books.get(i).getBookId());
            }
           }
           //throw exception if the publisher was not found
            if(found == false)
              throw new Exception("Please the library has no book published by this publisher");
          }
       catch (Exception e)
          {
           output(e.getMessage());
          }
    }
    /**
     * This method searches through the bookDetails array list for books of the searched
     * for genre, and outputs the details of all the books of that genre
     */
    public void searchBookByGenre()
```

20.



```
649     }
650     /**
651      * This method searches through the bookDetails array list for books of the searched
652      * for genre, and outputs the details of all the books of that genre
653      */
654     public void searchBookByGenre()
655     {
656        try
657           {
658            String searchedGenre = inputString("Enter the name of the genre you are searching for:");
659            boolean found = false;
660            for(int i = 0; i < books.size(); i ++)
661            {
662             //output the details of all the books of the inputted genre
663             if(this.books.get(i).getGenre().equalsIgnoreCase(searchedGenre))
664             {
665              found = true;
666              String foundGenre = this.books.get(i).getGenre();
667              String foundTitle = this.books.get(i).getBookTitle();
668              String foundAuthor = this.books.get(i).getAuthor();
669              String foundPublisher = this.books.get(i).getPublisher();
670              String foundCondition = this.books.get(i).getBookCondition();
671              String foundStatus = this.books.get(i).getStatus();
672              String foundId = this.books.get(i).getBookId();
673              output(foundGenre + "\t" + foundTitle + "\t" + foundAuthor + "\t" + foundPublisher + "\t" + foundCondition + "\t" + foundStatus + "\t"
674             }
675            }
676             if(found == false)
677               throw new Exception("Please the library has no book of this genre");
678           }
679        catch (Exception e)
680           {
681            output(e.getMessage());
682           }
683     }
684     /**
685      * This method searches through the studentDetails array list for students who have been fined and
686      * outputs the details of these students
687      */
688     public void searchStudentByFine()
```

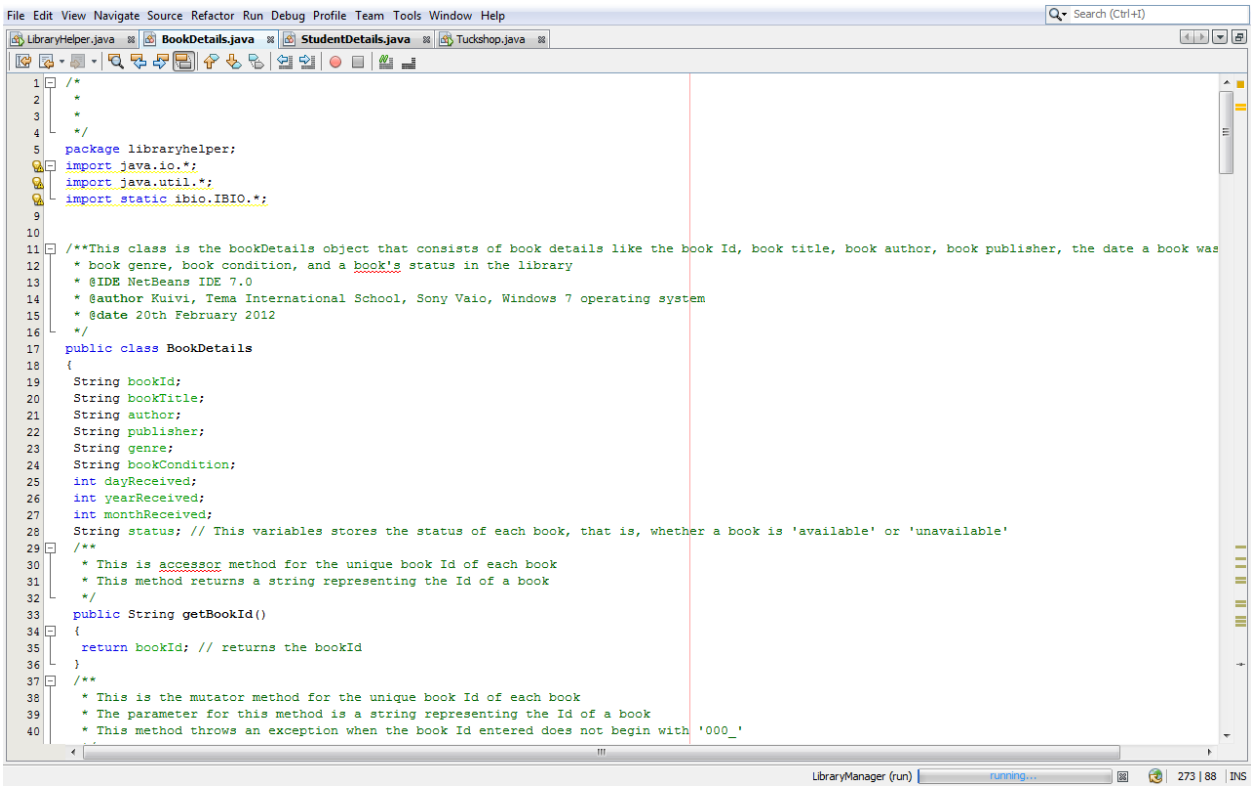
21.



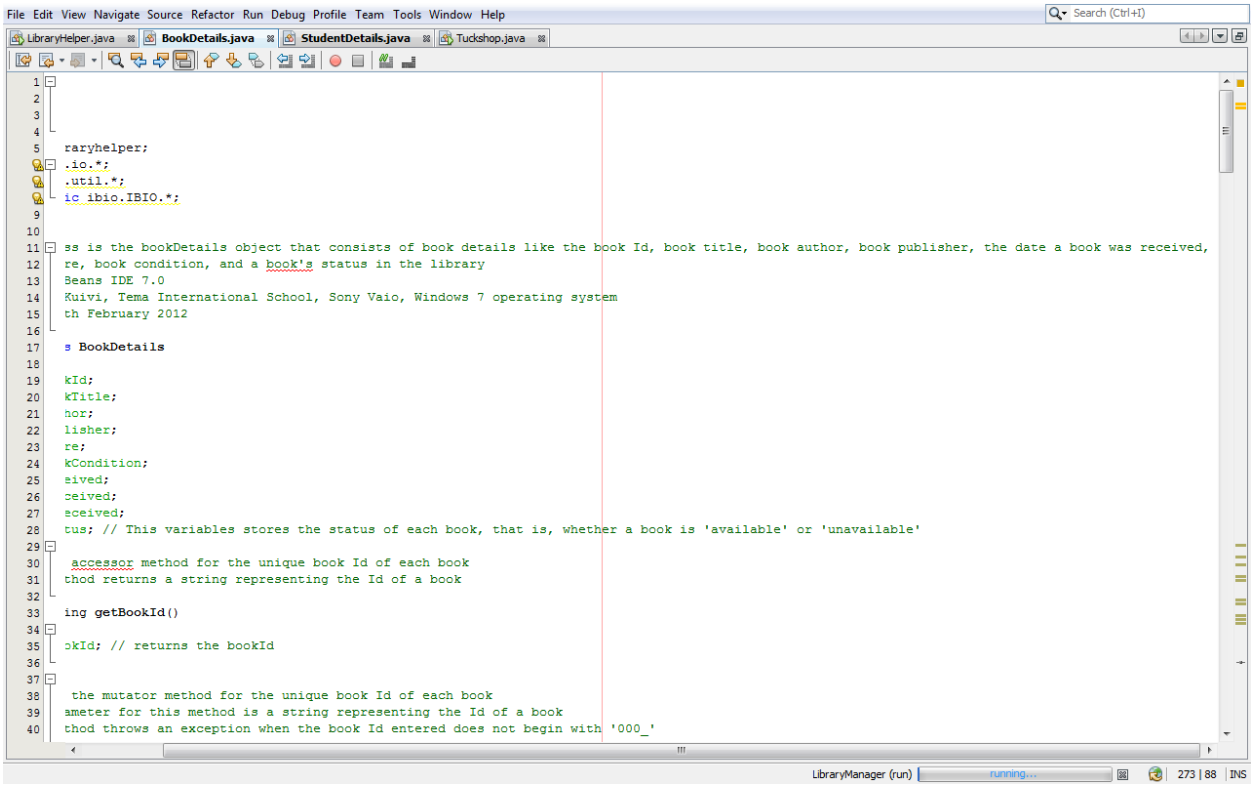
```
682           }
683     }
684     /**
685      * This method searches through the studentDetails array list for students who have been fined and
686      * outputs the details of these students
687      */
688     public void searchStudentByFine()
689     {
690        try
691           {
692            int found = -1;
693            for(int i = 0; i < student.size(); i ++)
694            {
695             //output the details of all students who have been fined by the library
696             if(this.student.get(i).getTotalFine() > 0)
697             {
698              found = 1;
699              String foundId = this.student.get(i).getStudentId();
700              String foundFirstName = this.student.get(i).getFirstName();
701              String foundLastName = this.student.get(i).getLastName();
702              int foundGrade = this.student.get(i).getGrade();
703              int foundTotalFine = this.student.get(i).getTotalFine();
704              output(foundId + "\t" + foundFirstName + "\t" + foundLastName + "\t" + foundGrade + "\t" + foundTotalFine);
705             }
706            }
707            //display message is no student has been fined by the library
708            if(found < 0)
709             output("No student in the library database has been fined");
710
711           }
712        catch (Exception e)
713           {
714            output(e.getMessage());
715           }
716     }
717  }
718
719
```

BookDetails Class

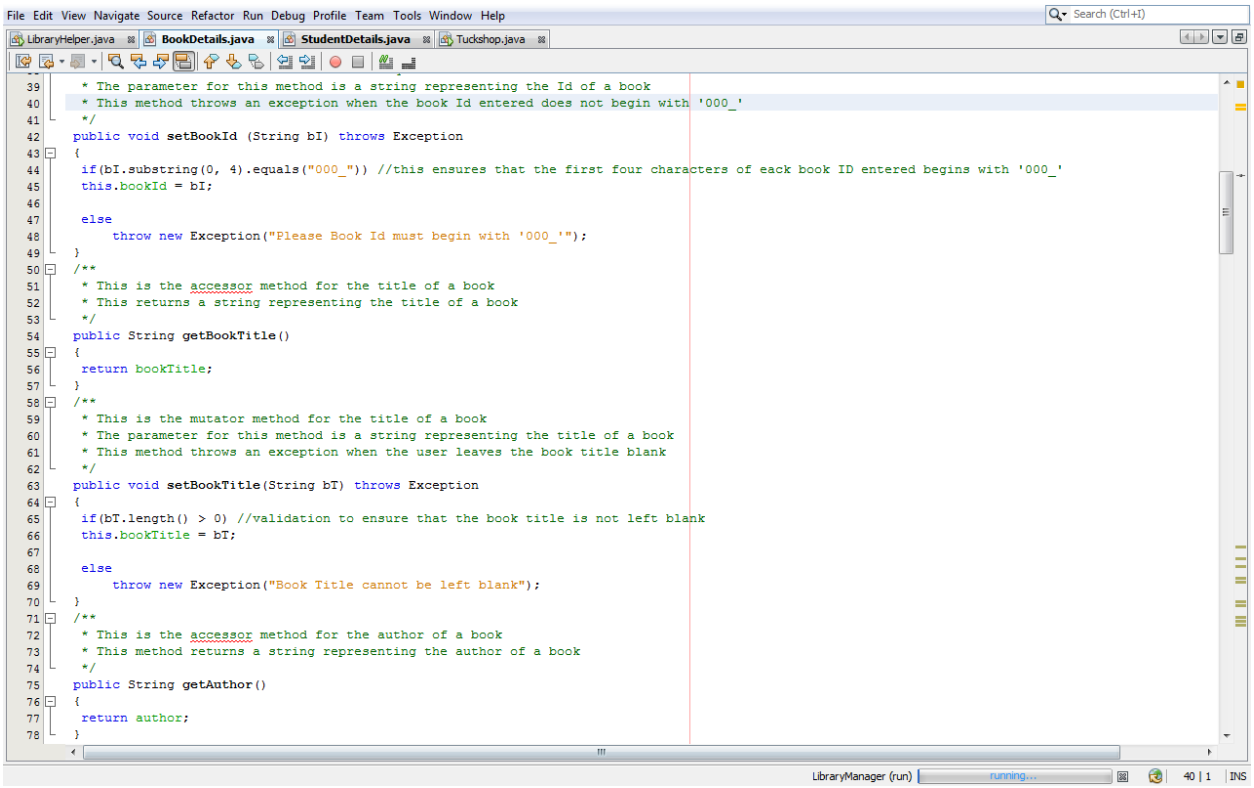
22.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
/*
*
*
*/
package libraryhelper;
import java.io.*;
import java.util.*;
import static ibio.IBIO.*;
/**This class is the bookDetails object that consists of book details like the book Id, book title, book author, book publisher, the date a book was
* book genre, book condition, and a book's status in the library
* @IDE NetBeans IDE 7.0
* @author Kuivi, Tema International School, Sony Vaio, Windows 7 operating system
* @date 20th February 2012
*/
public class BookDetails
{
String bookId;
String bookTitle;
String author;
String publisher;
String genre;
String bookCondition;
int dayReceived;
int yearReceived;
int monthReceived;
String status; // This variables stores the status of each book, that is, whether a book is 'available' or 'unavailable'
/**
* This is accessor method for the unique book Id of each book
* This method returns a string representing the Id of a book
*/
public String getBookId()
{
return bookId; // returns the bookId
}
/**
* This is the mutator method for the unique book Id of each book
* The parameter for this method is a string representing the Id of a book
* This method throws an exception when the book Id entered does not begin with '000_'
LibraryManager (run)
running...
273 | 88
INS

23.

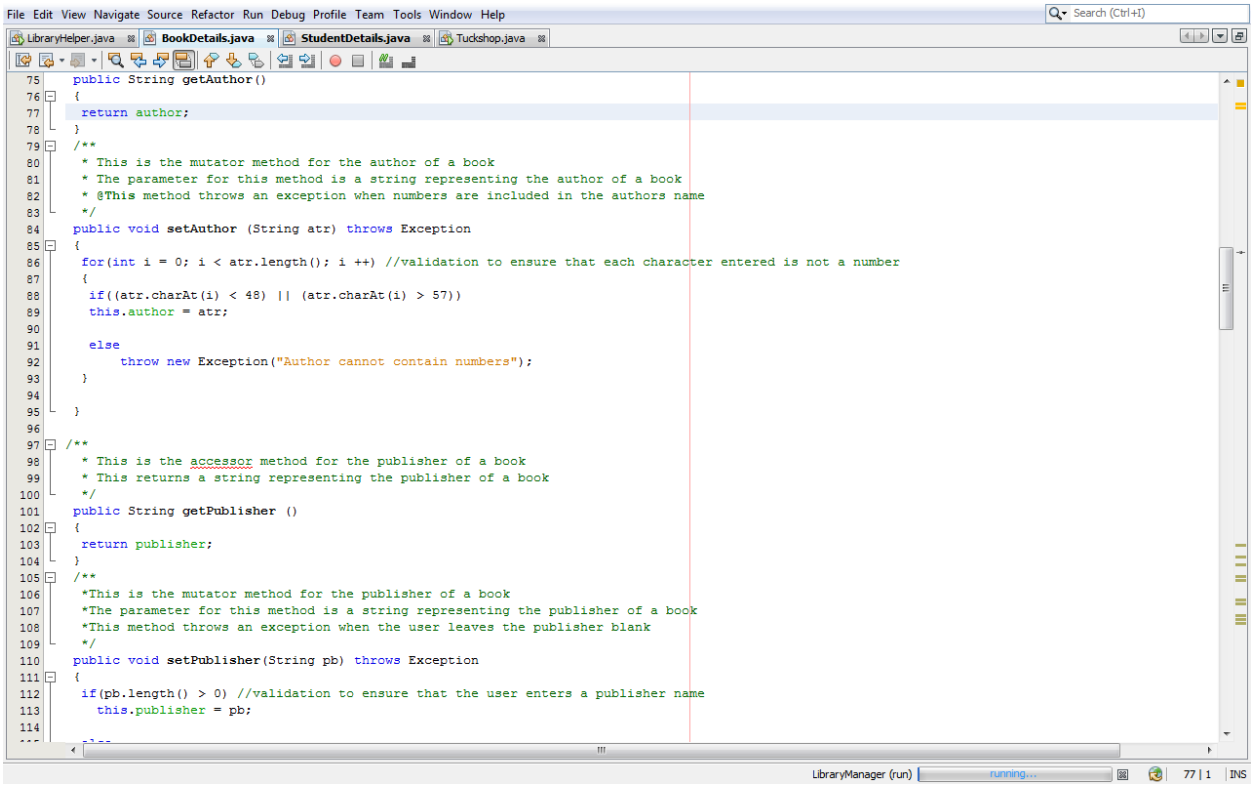

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
raryhelper;
.io.*;
.util.*;
ic ibio.IBIO.*;
ss is the bookDetails object that consists of book details like the book Id, book title, book author, book publisher, the date a book was received,
re, book condition, and a book's status in the library
Beans IDE 7.0
Kuivi, Tema International School, Sony Vaio, Windows 7 operating system
th February 2012
s BookDetails
kId;
kTitle;
hor;
lisher;
re;
kCondition;
eived;
ceived;
eceived;
tus; // This variables stores the status of each book, that is, whether a book is 'available' or 'unavailable'
accessor method for the unique book Id of each book
thod returns a string representing the Id of a book
ing getBookId()
okId; // returns the bookId
the mutator method for the unique book Id of each book
ameter for this method is a string representing the Id of a book
thod throws an exception when the book Id entered does not begin with '000_'
LibraryManager (run)
running...
273 | 88
INS

24.



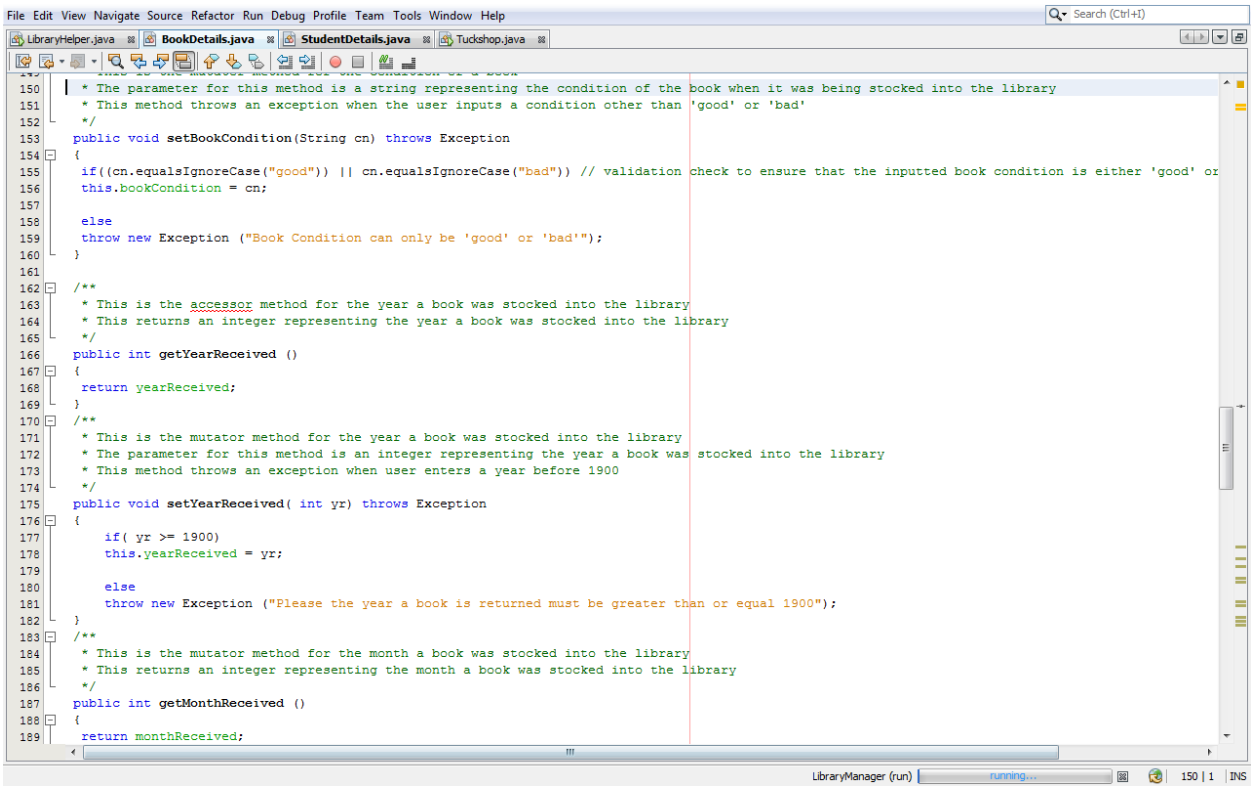
```
 * The parameter for this method is a string representing the Id of a book
 * This method throws an exception when the book Id entered does not begin with '000_'
 */
public void setBookId (String bI) throws Exception
{
 if(bI.substring(0, 4).equals("000_")) //this ensures that the first four characters of eack book ID entered begins with '000_'
 this.bookId = bI;

 else
     throw new Exception("Please Book Id must begin with '000_'");
}
/**
 * This is the accessor method for the title of a book
 * This returns a string representing the title of a book
 */
public String getBookTitle()
{
 return bookTitle;
}
/**
 * This is the mutator method for the title of a book
 * The parameter for this method is a string representing the title of a book
 * This method throws an exception when the user leaves the book title blank
 */
public void setBookTitle(String bT) throws Exception
{
 if(bT.length() > 0) //validation to ensure that the book title is not left blank
 this.bookTitle = bT;

 else
     throw new Exception("Book Title cannot be left blank");
}
/**
 * This is the accessor method for the author of a book
 * This method returns a string representing the author of a book
 */
public String getAuthor()
{
 return author;
}
```

25.



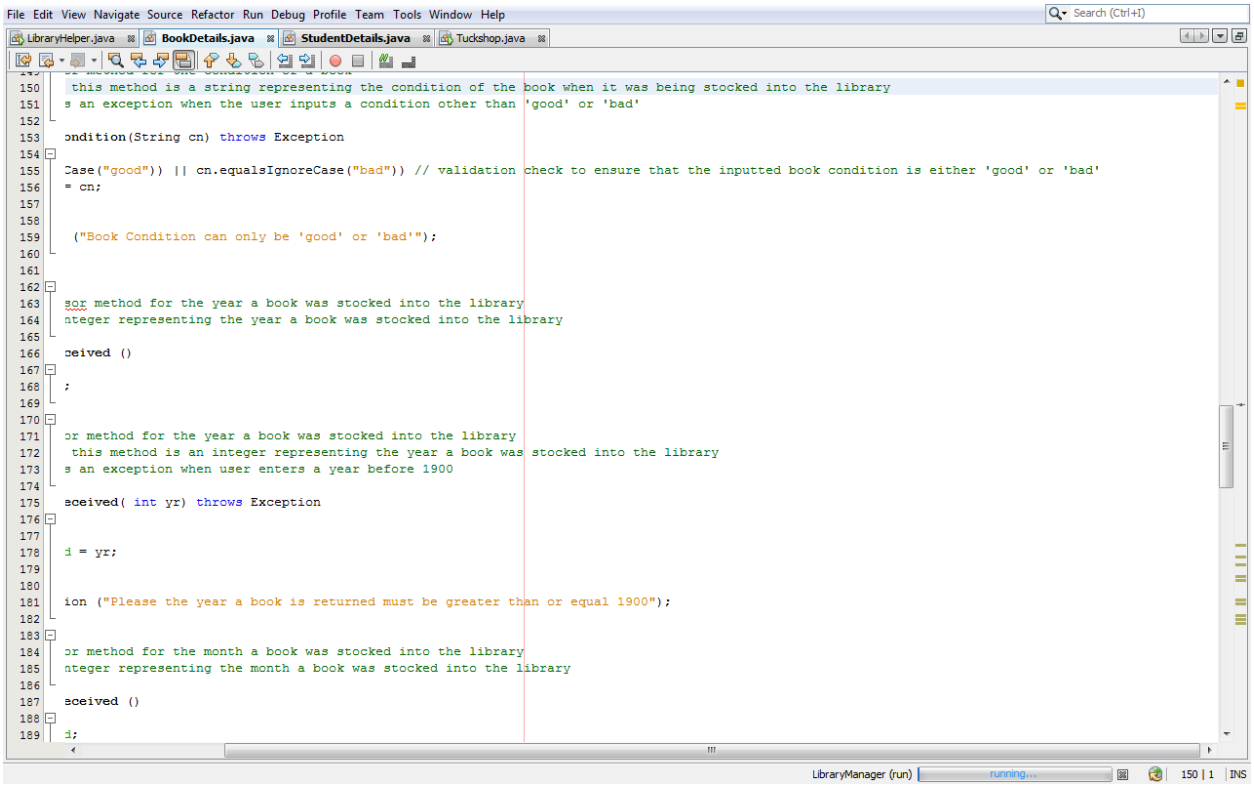
```
public String getAuthor()
{
 return author;
}
/**
 * This is the mutator method for the author of a book
 * The parameter for this method is a string representing the author of a book
 * @This method throws an exception when numbers are included in the authors name
 */
public void setAuthor (String atr) throws Exception
{
 for(int i = 0; i < atr.length(); i ++) //validation to ensure that each character entered is not a number
 {
  if((atr.charAt(i) < 48) || (atr.charAt(i) > 57))
  this.author = atr;

  else
      throw new Exception("Author cannot contain numbers");
 }

}

/**
 * This is the accessor method for the publisher of a book
 * This returns a string representing the publisher of a book
 */
public String getPublisher ()
{
 return publisher;
}
/**
 *This is the mutator method for the publisher of a book
 *The parameter for this method is a string representing the publisher of a book
 *This method throws an exception when the user leaves the publisher blank
 */
public void setPublisher(String pb) throws Exception
{
 if(pb.length() > 0) //validation to ensure that the user enters a publisher name
   this.publisher = pb;
```

26.



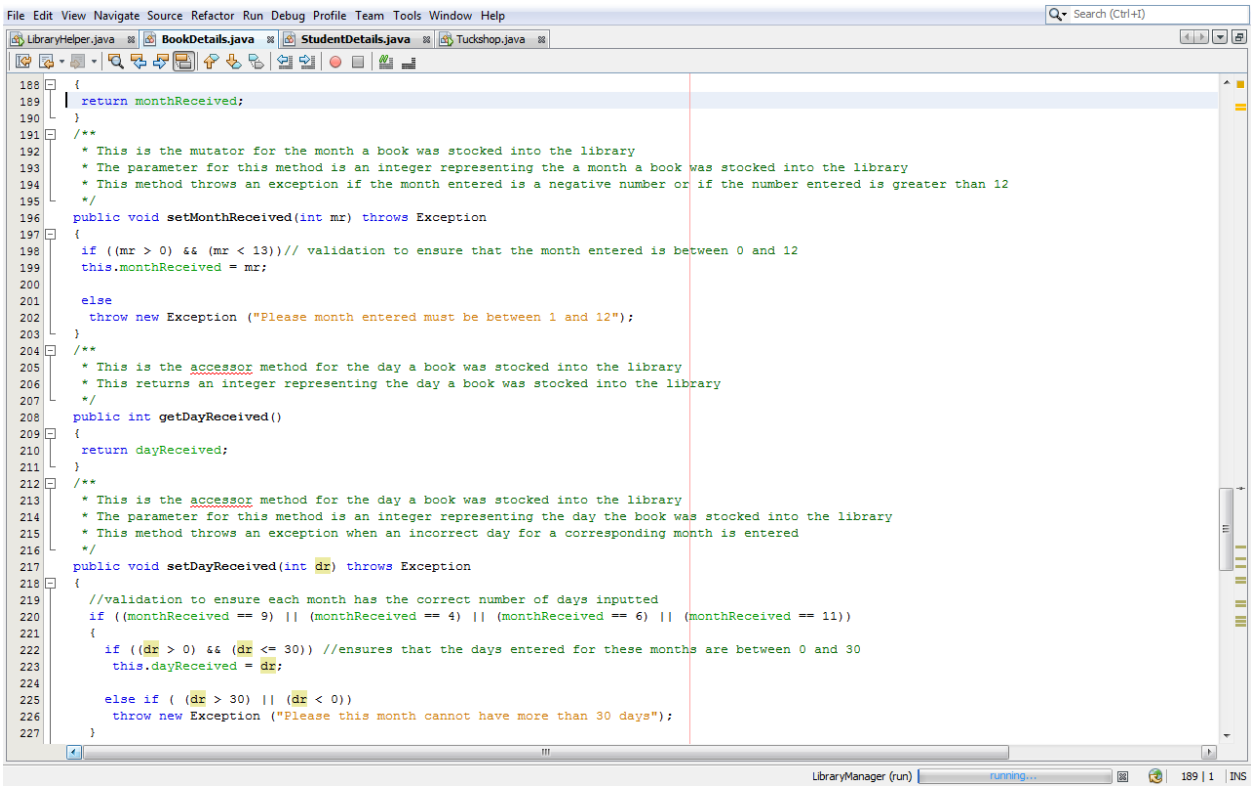
```
150     * The parameter for this method is a string representing the condition of the book when it was being stocked into the library
151     * This method throws an exception when the user inputs a condition other than 'good' or 'bad'
152     */
153    public void setBookCondition(String cn) throws Exception
154    {
155     if((cn.equalsIgnoreCase("good")) || cn.equalsIgnoreCase("bad")) // validation check to ensure that the inputted book condition is either 'good' or
156     this.bookCondition = cn;
157
158     else
159     throw new Exception ("Book Condition can only be 'good' or 'bad'");
160    }
161
162    /**
163     * This is the accessor method for the year a book was stocked into the library
164     * This returns an integer representing the year a book was stocked into the library
165     */
166    public int getYearReceived ()
167    {
168     return yearReceived;
169    }
170    /**
171     * This is the mutator method for the year a book was stocked into the library
172     * The parameter for this method is an integer representing the year a book was stocked into the library
173     * This method throws an exception when user enters a year before 1900
174     */
175    public void setYearReceived( int yr) throws Exception
176    {
177        if( yr >= 1900)
178        this.yearReceived = yr;
179
180        else
181        throw new Exception ("Please the year a book is returned must be greater than or equal 1900");
182    }
183    /**
184     * This is the mutator method for the month a book was stocked into the library
185     * This returns an integer representing the month a book was stocked into the library
186     */
187    public int getMonthReceived ()
188    {
189     return monthReceived;
```

27.



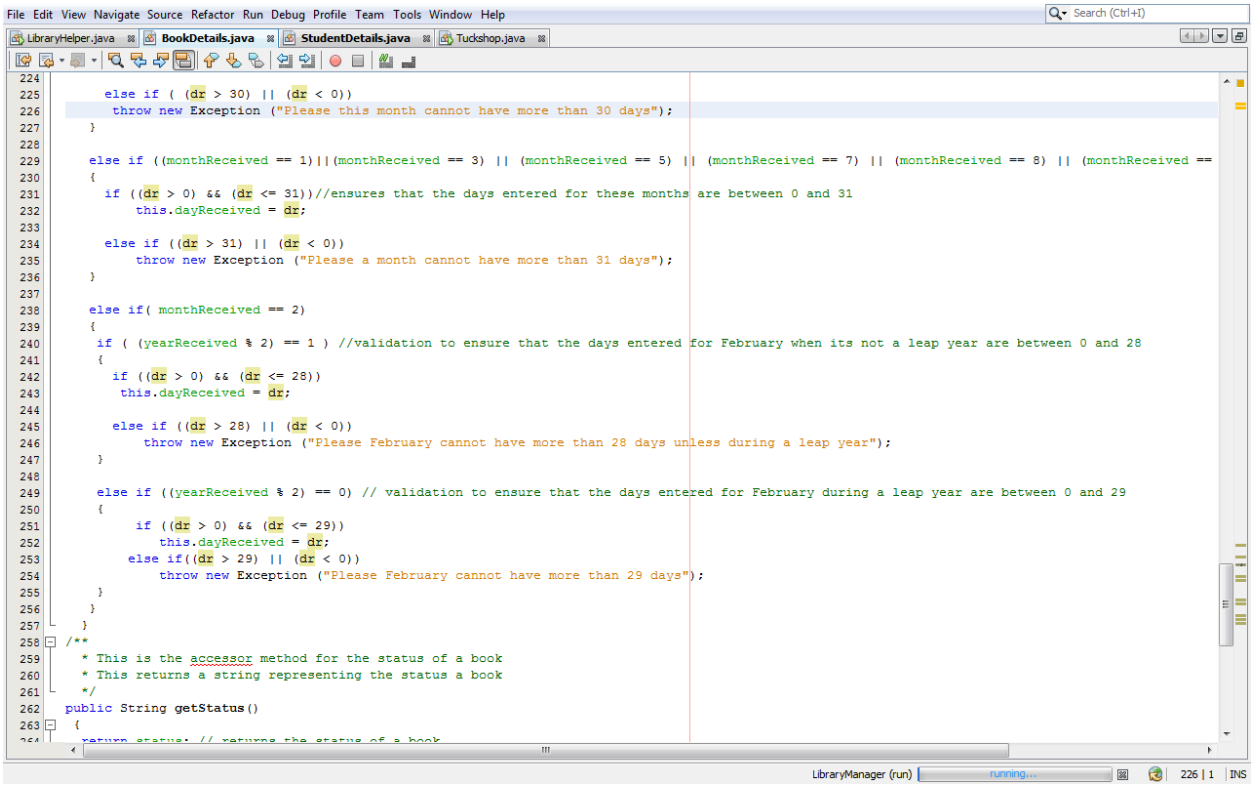
```
150     this method is a string representing the condition of the book when it was being stocked into the library
151    s an exception when the user inputs a condition other than 'good' or 'bad'
152
153    ondition(String cn) throws Exception
154
155    Case("good")) || cn.equalsIgnoreCase("bad")) // validation check to ensure that the inputted book condition is either 'good' or 'bad'
156    = cn;
157
158
159     ("Book Condition can only be 'good' or 'bad'");
160
161
162
163    sor method for the year a book was stocked into the library
164    nteger representing the year a book was stocked into the library
165
166    ceived ()
167
168    ;
169
170
171    or method for the year a book was stocked into the library
172     this method is an integer representing the year a book was stocked into the library
173    s an exception when user enters a year before 1900
174
175    sceived( int yr) throws Exception
176
177
178    i = yr;
179
180
181    ion ("Please the year a book is returned must be greater than or equal 1900");
182
183
184    or method for the month a book was stocked into the library
185    nteger representing the month a book was stocked into the library
186
187    sceived ()
188
189    i;
```

28.



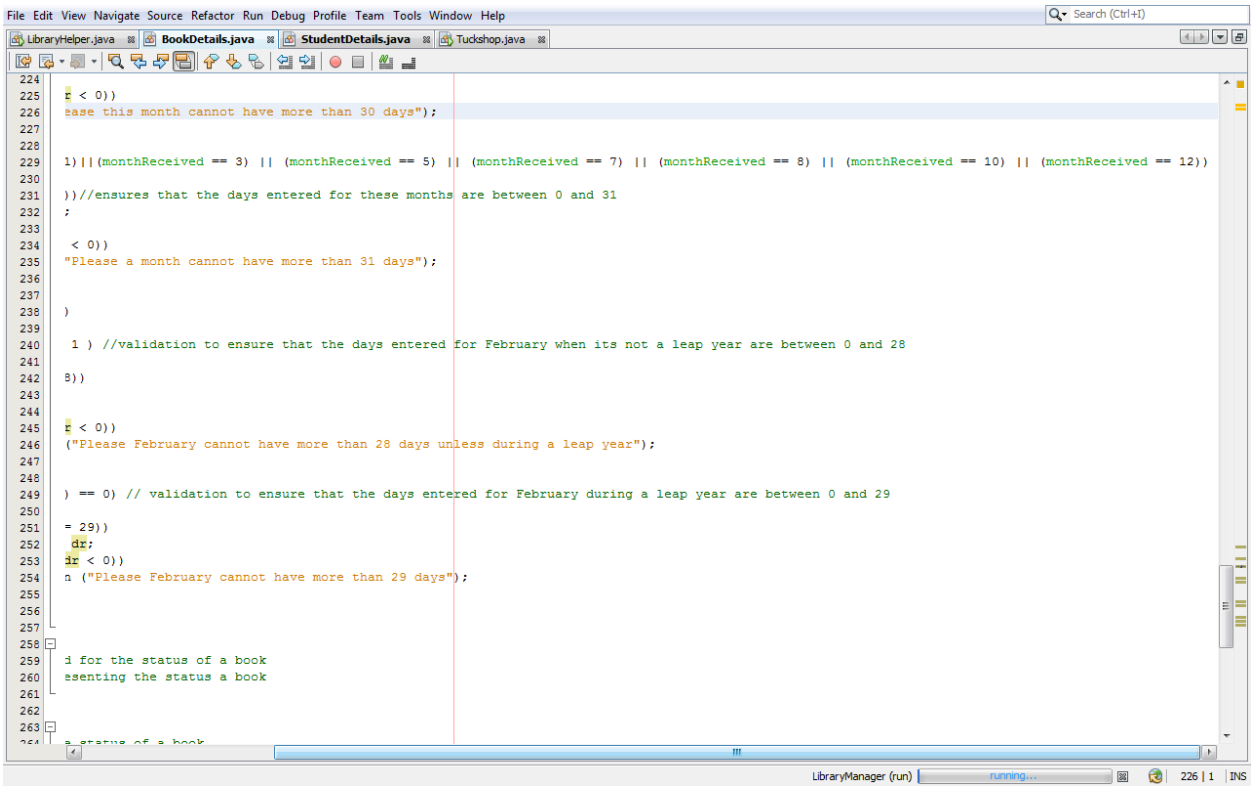
```
188  {
189   return monthReceived;
190  }
191  /**
192   * This is the mutator for the month a book was stocked into the library
193   * The parameter for this method is an integer representing the a month a book was stocked into the library
194   * This method throws an exception if the month entered is a negative number or if the number entered is greater than 12
195   */
196  public void setMonthReceived(int mr) throws Exception
197  {
198   if ((mr > 0) && (mr < 13))// validation to ensure that the month entered is between 0 and 12
199   this.monthReceived = mr;
200
201   else
202    throw new Exception ("Please month entered must be between 1 and 12");
203  }
204  /**
205   * This is the accessor method for the day a book was stocked into the library
206   * This returns an integer representing the day a book was stocked into the library
207   */
208  public int getDayReceived()
209  {
210   return dayReceived;
211  }
212  /**
213   * This is the accessor method for the day a book was stocked into the library
214   * The parameter for this method is an integer representing the day the book was stocked into the library
215   * This method throws an exception when an incorrect day for a corresponding month is entered
216   */
217  public void setDayReceived(int dr) throws Exception
218  {
219    //validation to ensure each month has the correct number of days inputted
220    if ((monthReceived == 9) || (monthReceived == 4) || (monthReceived == 6) || (monthReceived == 11))
221    {
222      if ((dr > 0) && (dr <= 30)) //ensures that the days entered for these months are between 0 and 30
223       this.dayReceived = dr;
224
225      else if ( (dr > 30) || (dr < 0))
226       throw new Exception ("Please this month cannot have more than 30 days");
227    }
```

29.

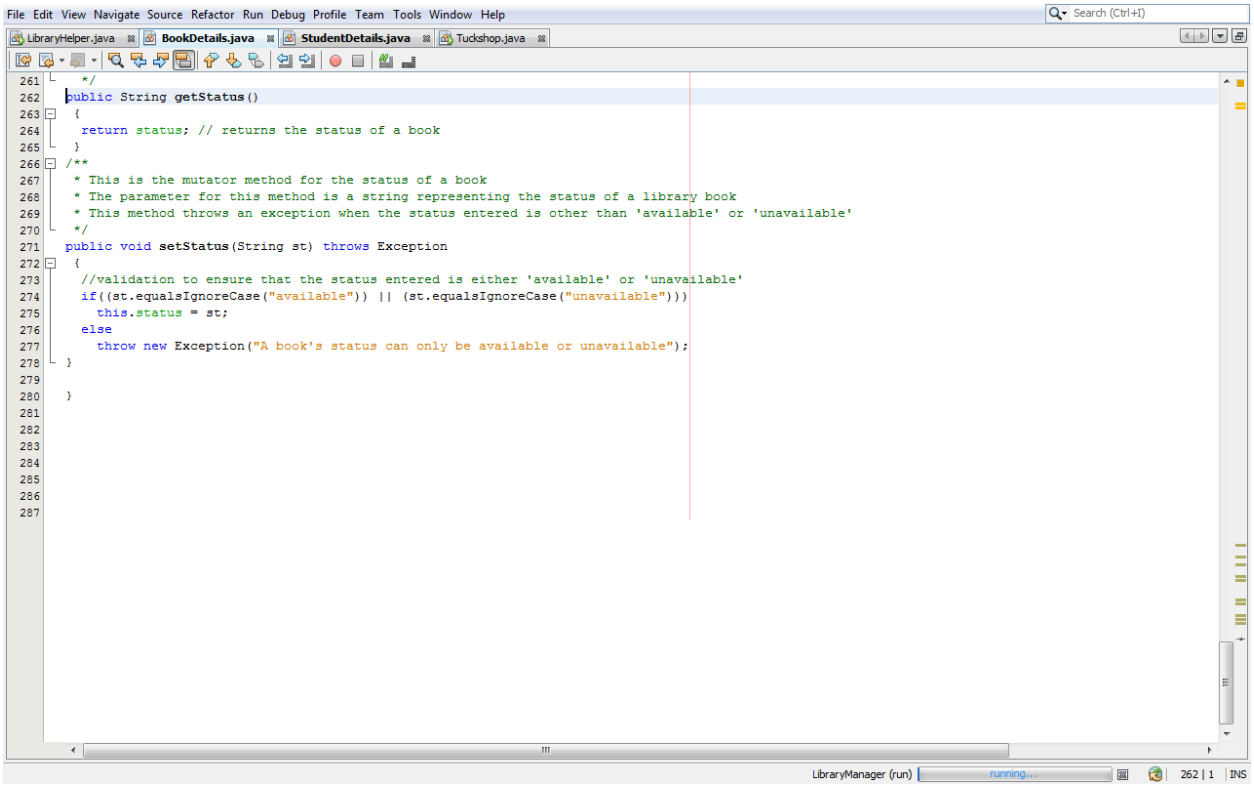


```
224
225      else if ( (dr > 30) || (dr < 0))
226       throw new Exception ("Please this month cannot have more than 30 days");
227    }
228
229    else if ((monthReceived == 1)||(monthReceived == 3) || (monthReceived == 5) || (monthReceived == 7) || (monthReceived == 8) || (monthReceived ==
230    {
231      if ((dr > 0) && (dr <= 31))//ensures that the days entered for these months are between 0 and 31
232          this.dayReceived = dr;
233
234      else if ((dr > 31) || (dr < 0))
235          throw new Exception ("Please a month cannot have more than 31 days");
236    }
237
238    else if( monthReceived == 2)
239    {
240     if ( (yearReceived % 2) == 1 ) //validation to ensure that the days entered for February when its not a leap year are between 0 and 28
241     {
242       if ((dr > 0) && (dr <= 28))
243        this.dayReceived = dr;
244
245       else if ((dr > 28) || (dr < 0))
246           throw new Exception ("Please February cannot have more than 28 days unless during a leap year");
247     }
248
249     else if ((yearReceived % 2) == 0) // validation to ensure that the days entered for February during a leap year are between 0 and 29
250     {
251          if ((dr > 0) && (dr <= 29))
252             this.dayReceived = dr;
253         else if((dr > 29) || (dr < 0))
254             throw new Exception ("Please February cannot have more than 29 days");
255     }
256    }
257   }
258  /**
259   * This is the accessor method for the status of a book
260   * This returns a string representing the status a book
261   */
262  public String getStatus()
263  {
264   return status; // returns the status of a book
```

30.

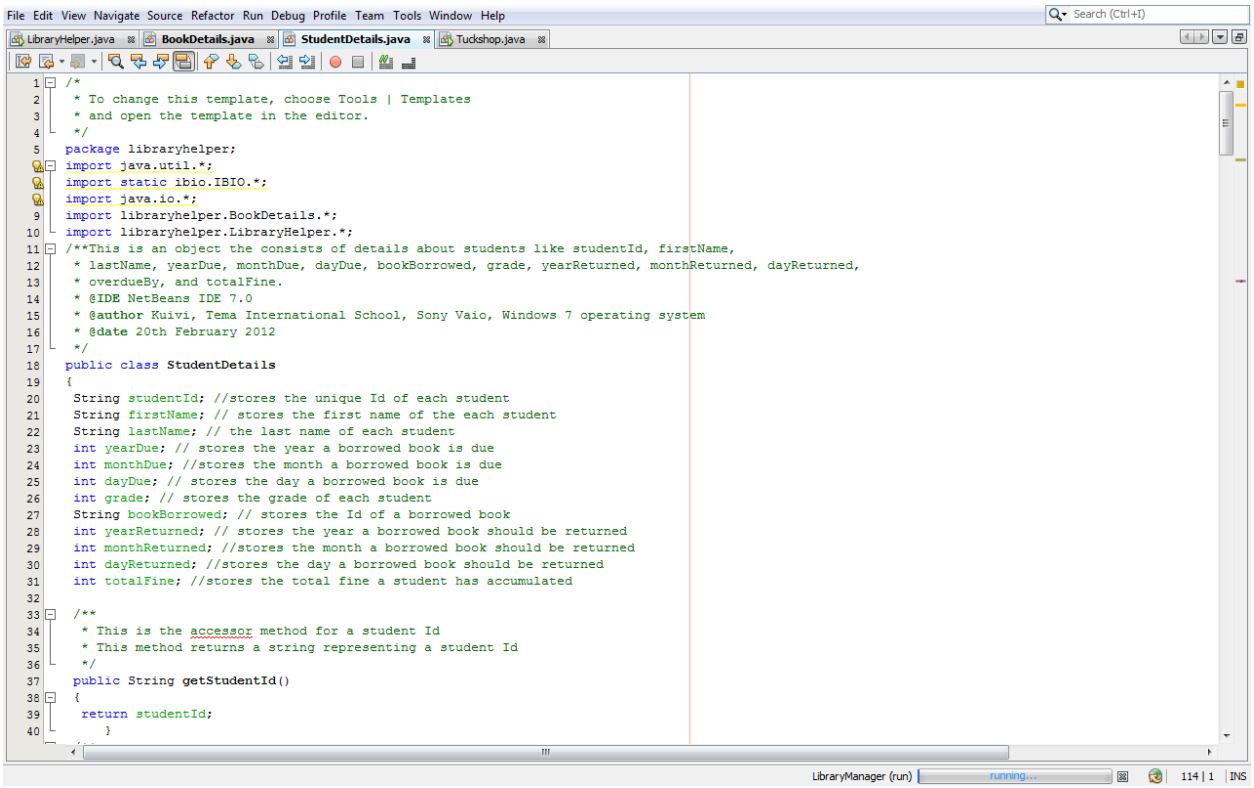

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
r < 0))
ease this month cannot have more than 30 days");
1)||(monthReceived == 3) || (monthReceived == 5) || (monthReceived == 7) || (monthReceived == 8) || (monthReceived == 10) || (monthReceived == 12))
))//ensures that the days entered for these months are between 0 and 31
< 0))
"Please a month cannot have more than 31 days");
1) //validation to ensure that the days entered for February when its not a leap year are between 0 and 28
8))
r < 0))
("Please February cannot have more than 28 days unless during a leap year");
) == 0) // validation to ensure that the days entered for February during a leap year are between 0 and 29
= 29))
dr;
ir < 0))
n ("Please February cannot have more than 29 days");
d for the status of a book
esenting the status a book
LibraryManager (run)
running...
226 | 1
INS

31.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
*/
public String getStatus()
{
return status; // returns the status of a book
}
/**
* This is the mutator method for the status of a book
* The parameter for this method is a string representing the status of a library book
* This method throws an exception when the status entered is other than 'available' or 'unavailable'
*/
public void setStatus(String st) throws Exception
{
//validation to ensure that the status entered is either 'available' or 'unavailable'
if((st.equalsIgnoreCase("available")) || (st.equalsIgnoreCase("unavailable")))
this.status = st;
else
throw new Exception("A book's status can only be available or unavailable");
}
}
LibraryManager (run)
running...
262 | 1
INS

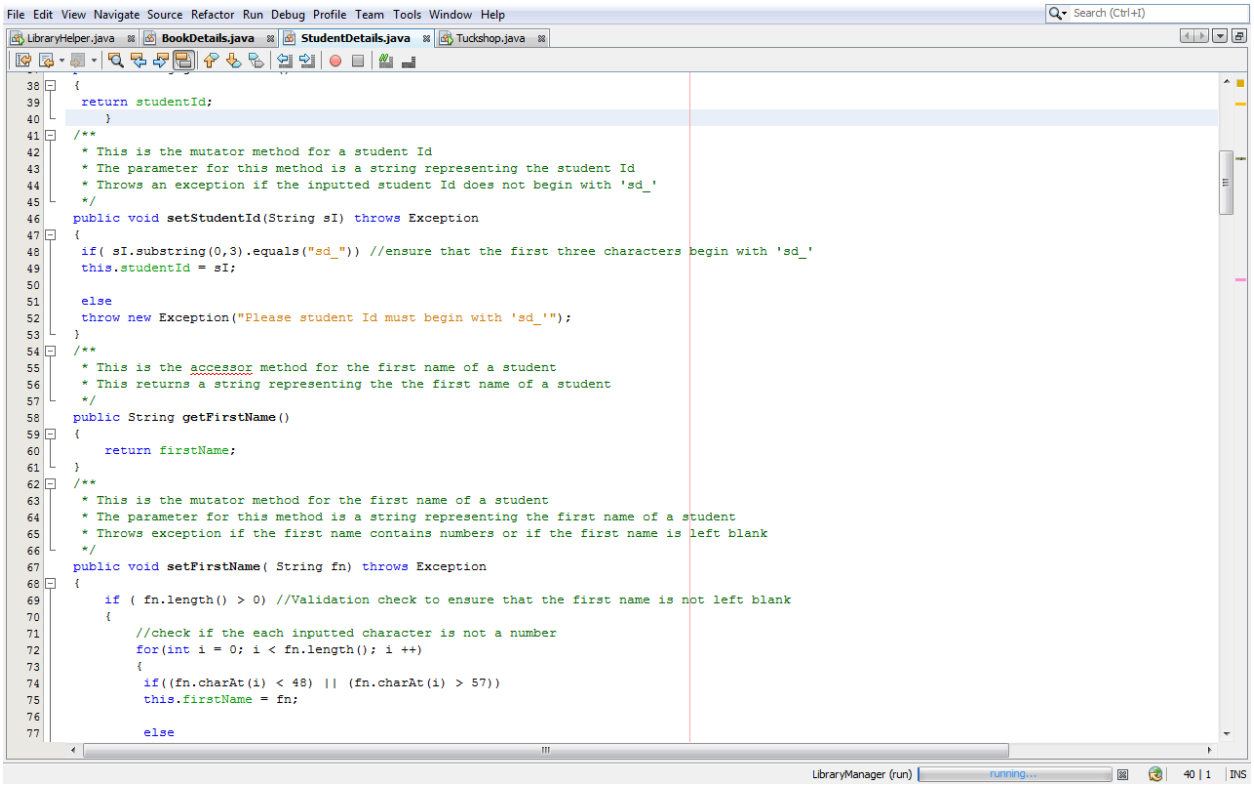
StudentDetails class

32.



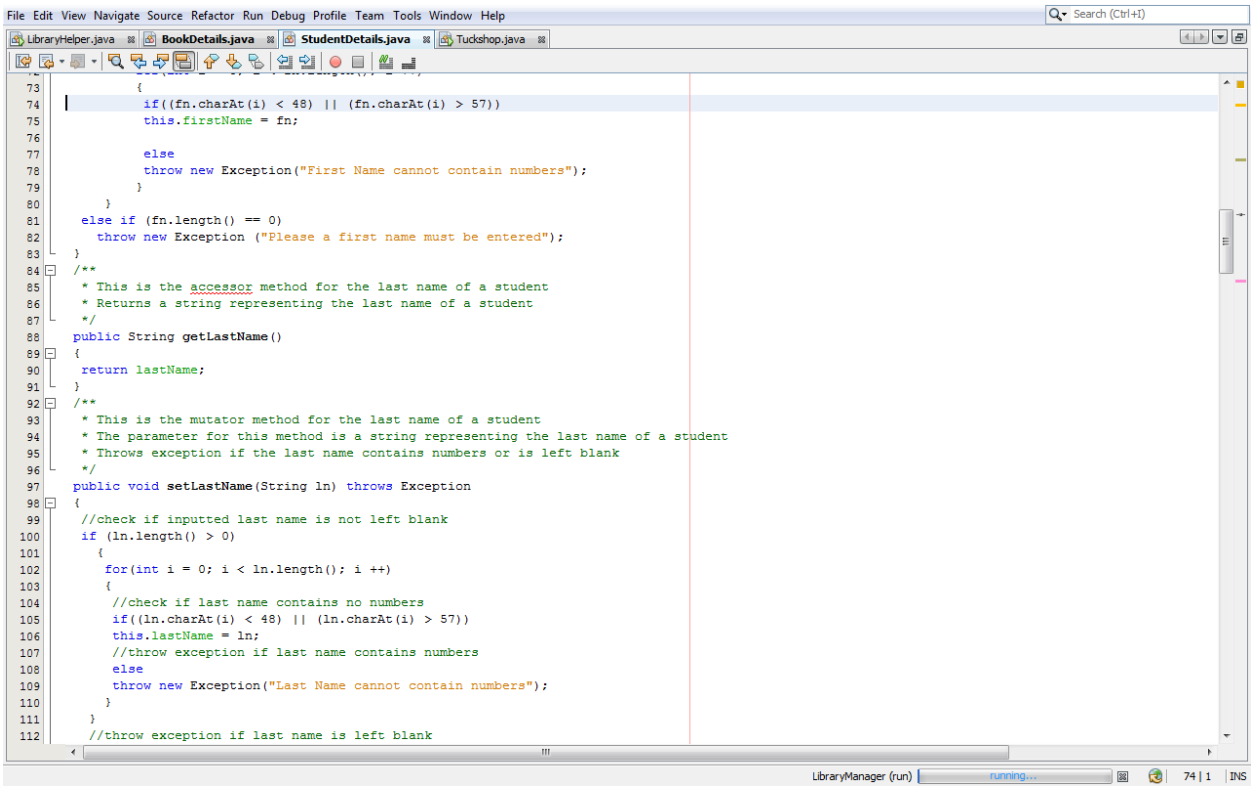
```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package libraryhelper;
import java.util.*;
import static ibio.IBIO.*;
import java.io.*;
import libraryhelper.BookDetails.*;
import libraryhelper.LibraryHelper.*;
/**This is an object the consists of details about students like studentId, firstName,
 * lastName, yearDue, monthDue, dayDue, bookBorrowed, grade, yearReturned, monthReturned, dayReturned,
 * overdueBy, and totalFine.
 * @IDE NetBeans IDE 7.0
 * @author Kuivi, Tema International School, Sony Vaio, Windows 7 operating system
 * @date 20th February 2012
 */
public class StudentDetails
{
 String studentId; //stores the unique Id of each student
 String firstName; // stores the first name of the each student
 String lastName; // the last name of each student
 int yearDue; // stores the year a borrowed book is due
 int monthDue; //stores the month a borrowed book is due
 int dayDue; // stores the day a borrowed book is due
 int grade; // stores the grade of each student
 String bookBorrowed; // stores the Id of a borrowed book
 int yearReturned; // stores the year a borrowed book should be returned
 int monthReturned; //stores the month a borrowed book should be returned
 int dayReturned; //stores the day a borrowed book should be returned
 int totalFine; //stores the total fine a student has accumulated

 /**
  * This is the accessor method for a student Id
  * This method returns a string representing a student Id
  */
 public String getStudentId()
 {
  return studentId;
     }
```

33.



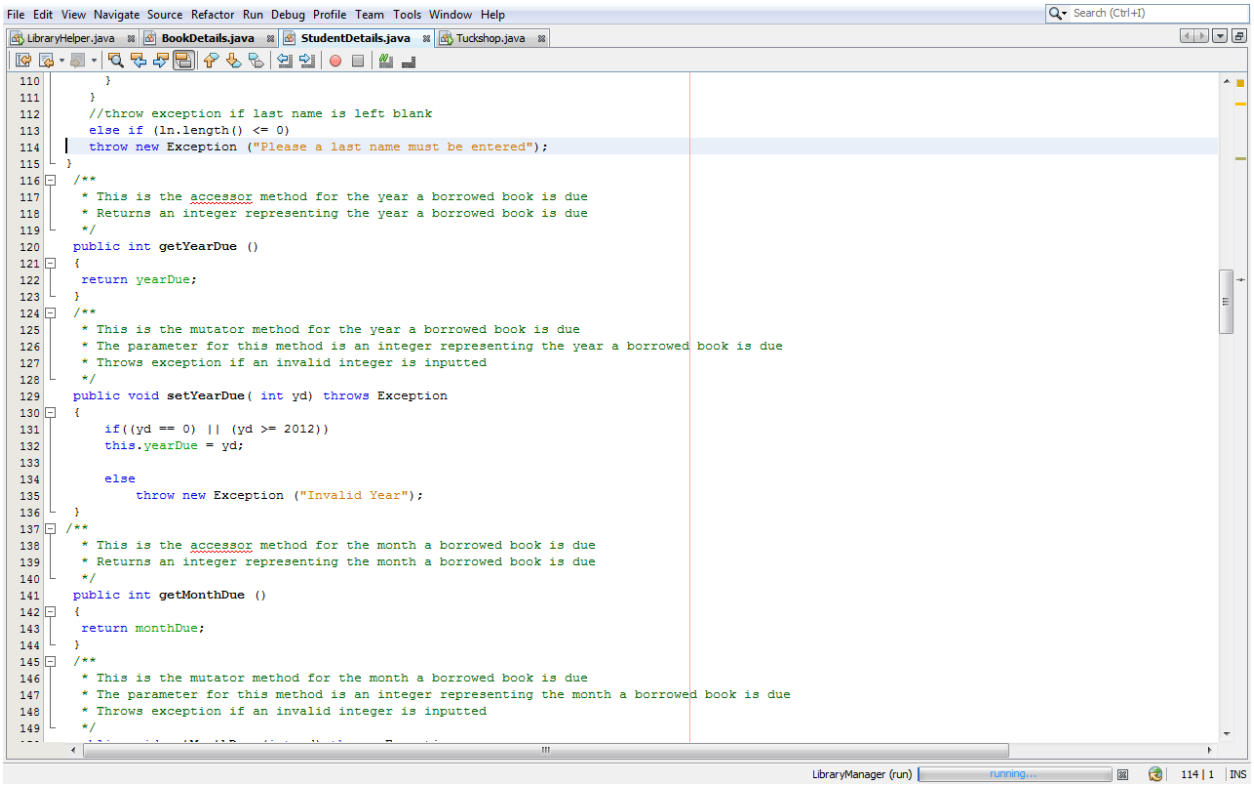
```
 {
  return studentId;
     }
 /**
  * This is the mutator method for a student Id
  * The parameter for this method is a string representing the student Id
  * Throws an exception if the inputted student Id does not begin with 'sd_'
  */
 public void setStudentId(String sI) throws Exception
 {
  if( sI.substring(0,3).equals("sd_")) //ensure that the first three characters begin with 'sd_'
  this.studentId = sI;

  else
  throw new Exception("Please student Id must begin with 'sd_'");
 }
 /**
  * This is the accessor method for the first name of a student
  * This returns a string representing the the first name of a student
  */
 public String getFirstName()
 {
     return firstName;
 }
 /**
  * This is the mutator method for the first name of a student
  * The parameter for this method is a string representing the first name of a student
  * Throws exception if the first name contains numbers or if the first name is left blank
  */
 public void setFirstName( String fn) throws Exception
 {
     if ( fn.length() > 0) //Validation check to ensure that the first name is not left blank
     {
         //check if the each inputted character is not a number
         for(int i = 0; i < fn.length(); i ++)
         {
          if((fn.charAt(i) < 48) || (fn.charAt(i) > 57))
          this.firstName = fn;

          else
```

34.



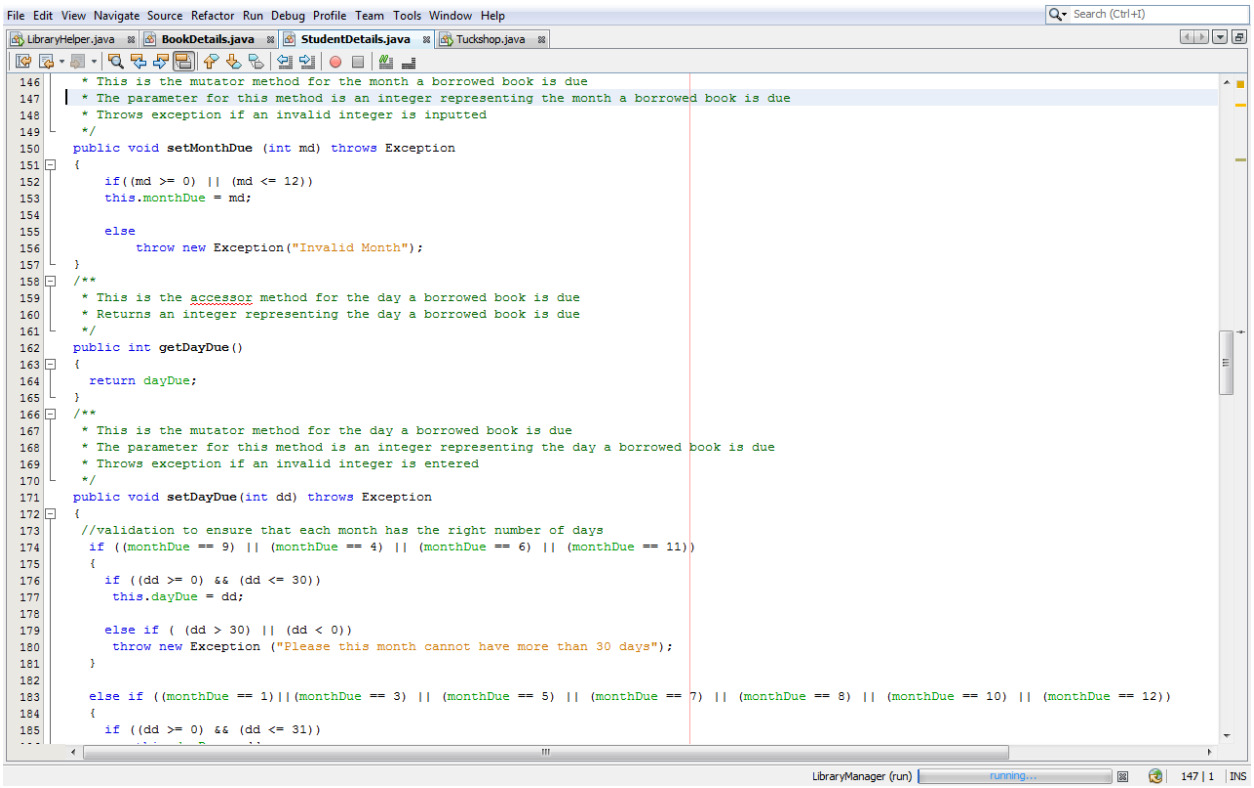
```
73          {
74           if((fn.charAt(i) < 48) || (fn.charAt(i) > 57))
75           this.firstName = fn;
76
77           else
78           throw new Exception("First Name cannot contain numbers");
79          }
80       }
81     else if (fn.length() == 0)
82       throw new Exception ("Please a first name must be entered");
83   }
84   /**
85    * This is the accessor method for the last name of a student
86    * Returns a string representing the last name of a student
87    */
88   public String getLastName()
89   {
90    return lastName;
91   }
92   /**
93    * This is the mutator method for the last name of a student
94    * The parameter for this method is a string representing the last name of a student
95    * Throws exception if the last name contains numbers or is left blank
96    */
97   public void setLastName(String ln) throws Exception
98   {
99    //check if inputted last name is not left blank
100   if (ln.length() > 0)
101     {
102      for(int i = 0; i < ln.length(); i ++)
103      {
104       //check if last name contains no numbers
105       if((ln.charAt(i) < 48) || (ln.charAt(i) > 57))
106       this.lastName = ln;
107       //throw exception if last name contains numbers
108       else
109       throw new Exception("Last Name cannot contain numbers");
110      }
111     }
112    //throw exception if last name is left blank
```

35.



```
110      }
111    }
112    //throw exception if last name is left blank
113    else if (ln.length() <= 0)
114    throw new Exception ("Please a last name must be entered");
115  }
116  /**
117   * This is the accessor method for the year a borrowed book is due
118   * Returns an integer representing the year a borrowed book is due
119   */
120  public int getYearDue ()
121  {
122   return yearDue;
123  }
124  /**
125   * This is the mutator method for the year a borrowed book is due
126   * The parameter for this method is an integer representing the year a borrowed book is due
127   * Throws exception if an invalid integer is inputted
128   */
129  public void setYearDue( int yd) throws Exception
130  {
131      if((yd == 0) || (yd >= 2012))
132      this.yearDue = yd;
133
134      else
135          throw new Exception ("Invalid Year");
136  }
137 /**
138   * This is the accessor method for the month a borrowed book is due
139   * Returns an integer representing the month a borrowed book is due
140   */
141  public int getMonthDue ()
142  {
143   return monthDue;
144  }
145  /**
146   * This is the mutator method for the month a borrowed book is due
147   * The parameter for this method is an integer representing the month a borrowed book is due
148   * Throws exception if an invalid integer is inputted
149   */
```

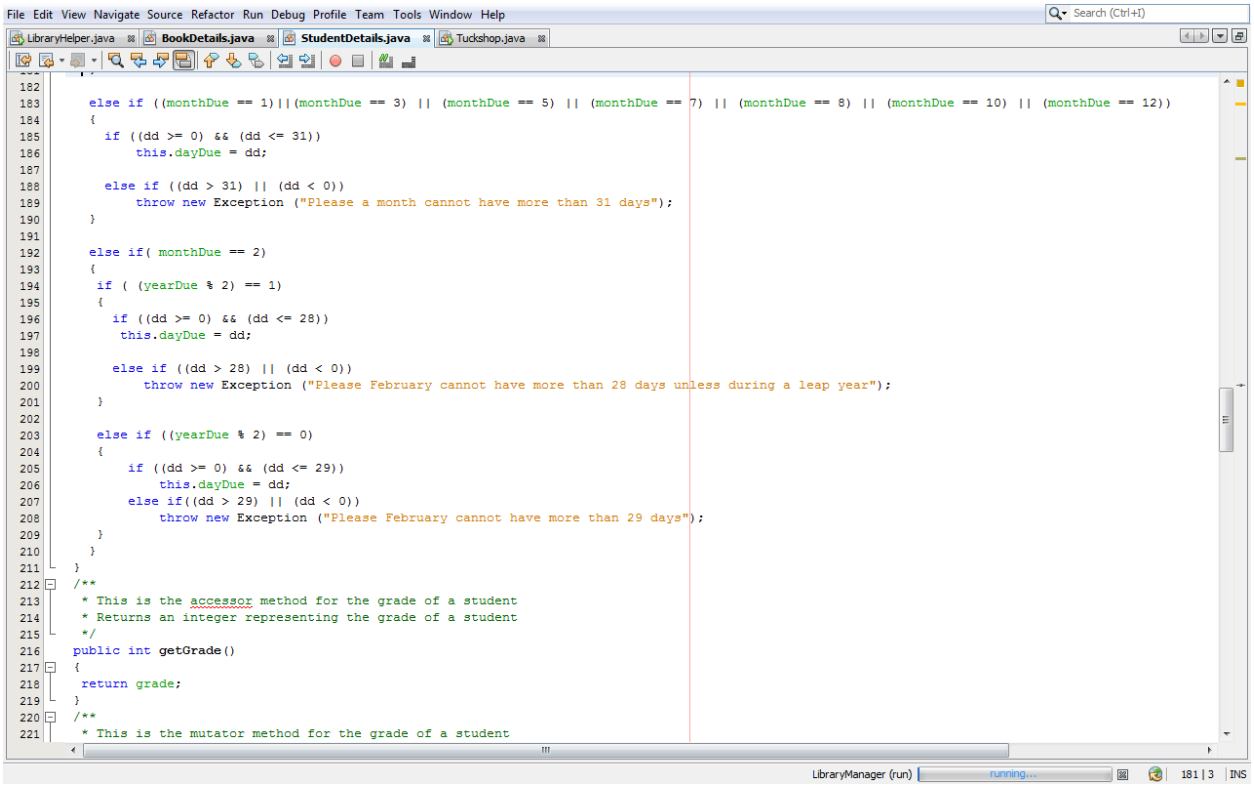
36.



```
146     * This is the mutator method for the month a borrowed book is due
147     * The parameter for this method is an integer representing the month a borrowed book is due
148     * Throws exception if an invalid integer is inputted
149     */
150    public void setMonthDue (int md) throws Exception
151    {
152        if((md >= 0) || (md <= 12))
153        this.monthDue = md;
154
155        else
156            throw new Exception("Invalid Month");
157    }
158    /**
159     * This is the accessor method for the day a borrowed book is due
160     * Returns an integer representing the day a borrowed book is due
161     */
162    public int getDayDue()
163    {
164      return dayDue;
165    }
166    /**
167     * This is the mutator method for the day a borrowed book is due
168     * The parameter for this method is an integer representing the day a borrowed book is due
169     * Throws exception if an invalid integer is entered
170     */
171    public void setDayDue(int dd) throws Exception
172    {
173     //validation to ensure that each month has the right number of days
174      if ((monthDue == 9) || (monthDue == 4) || (monthDue == 6) || (monthDue == 11))
175      {
176        if ((dd >= 0) && (dd <= 30))
177         this.dayDue = dd;
178
179        else if ( (dd > 30) || (dd < 0))
180         throw new Exception ("Please this month cannot have more than 30 days");
181      }
182
183      else if ((monthDue == 1)||(monthDue == 3) || (monthDue == 5) || (monthDue == 7) || (monthDue == 8) || (monthDue == 10) || (monthDue == 12))
184      {
185        if ((dd >= 0) && (dd <= 31))
```

LibraryManager (run) running... 147 | 1 INS

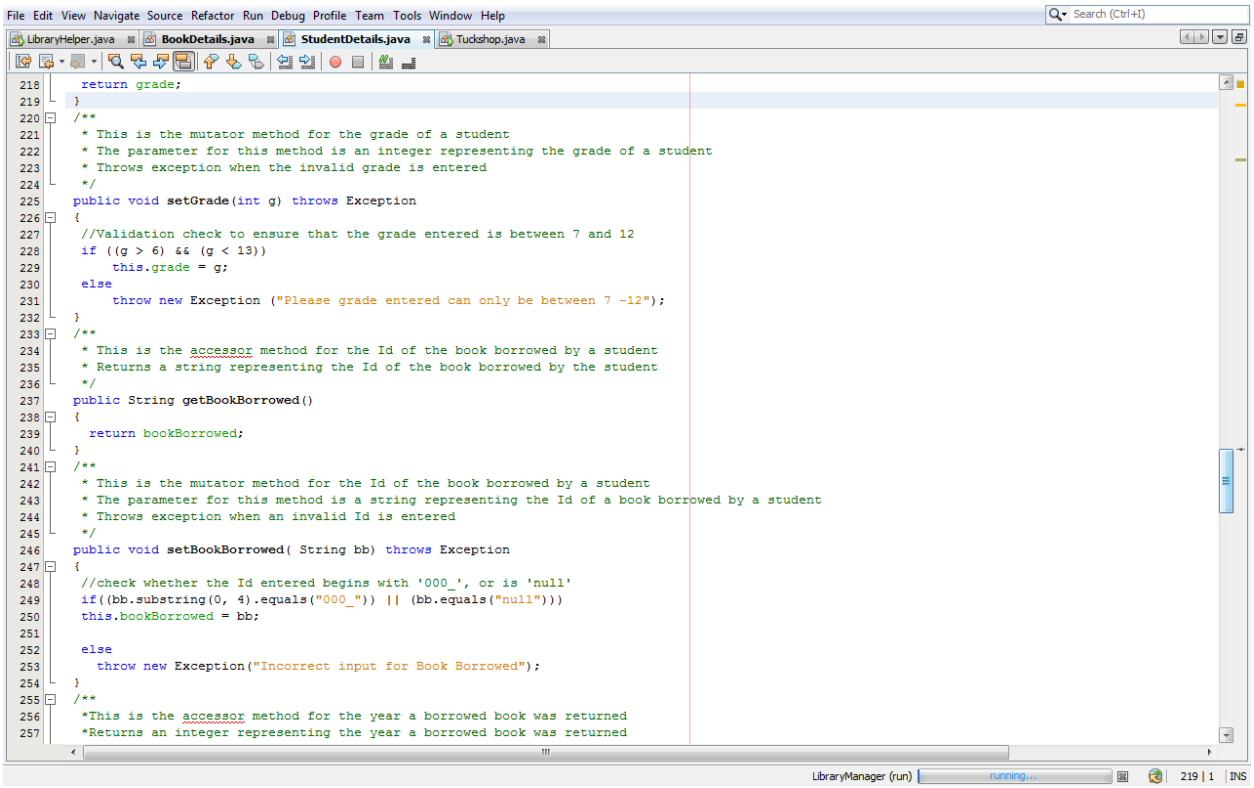
37.



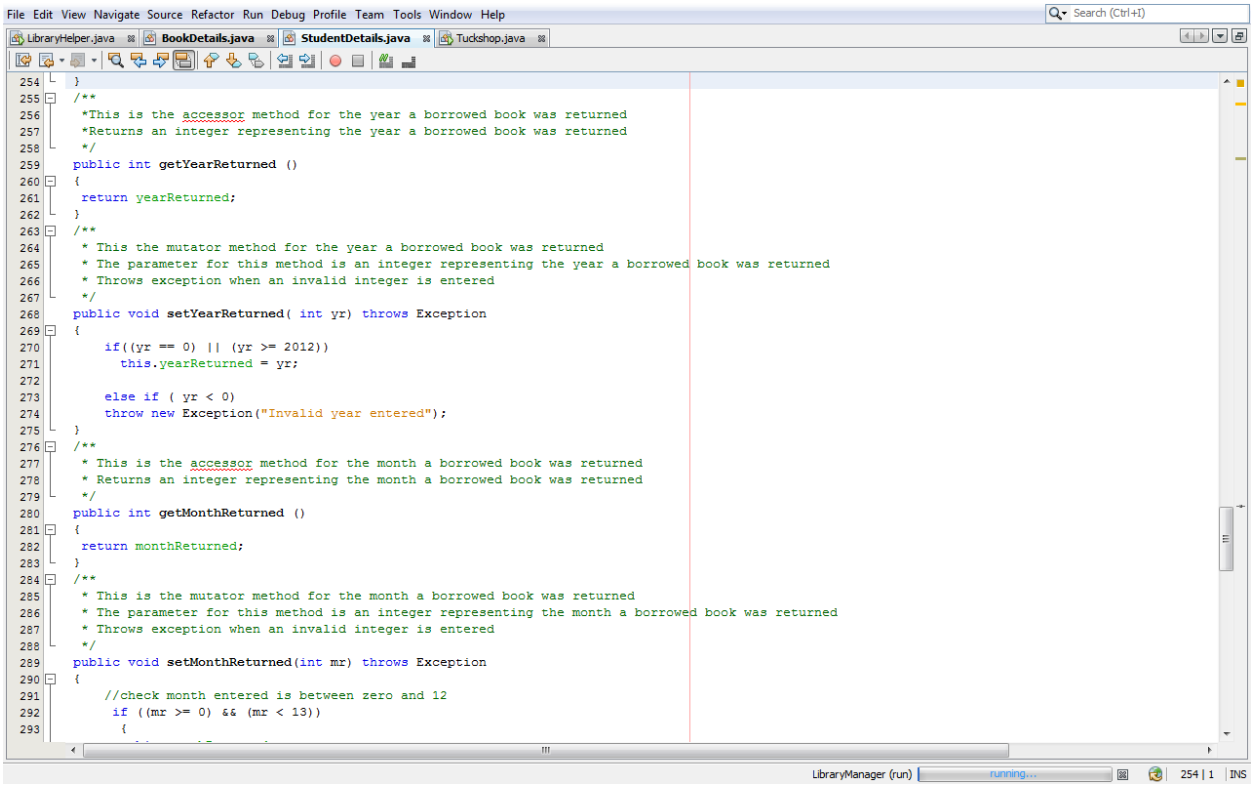
```
182
183      else if ((monthDue == 1)||(monthDue == 3) || (monthDue == 5) || (monthDue == 7) || (monthDue == 8) || (monthDue == 10) || (monthDue == 12))
184      {
185        if ((dd >= 0) && (dd <= 31))
186            this.dayDue = dd;
187
188        else if ((dd > 31) || (dd < 0))
189            throw new Exception ("Please a month cannot have more than 31 days");
190      }
191
192      else if( monthDue == 2)
193      {
194       if ( (yearDue % 2) == 1)
195       {
196         if ((dd >= 0) && (dd <= 28))
197          this.dayDue = dd;
198
199         else if ((dd > 28) || (dd < 0))
200             throw new Exception ("Please February cannot have more than 28 days unless during a leap year");
201       }
202
203       else if ((yearDue % 2) == 0)
204       {
205           if ((dd >= 0) && (dd <= 29))
206               this.dayDue = dd;
207           else if((dd > 29) || (dd < 0))
208               throw new Exception ("Please February cannot have more than 29 days");
209       }
210      }
211    }
212    /**
213     * This is the accessor method for the grade of a student
214     * Returns an integer representing the grade of a student
215     */
216    public int getGrade()
217    {
218     return grade;
219    }
220    /**
221     * This is the mutator method for the grade of a student
```

LibraryManager (run) running... 181 | 3 INS

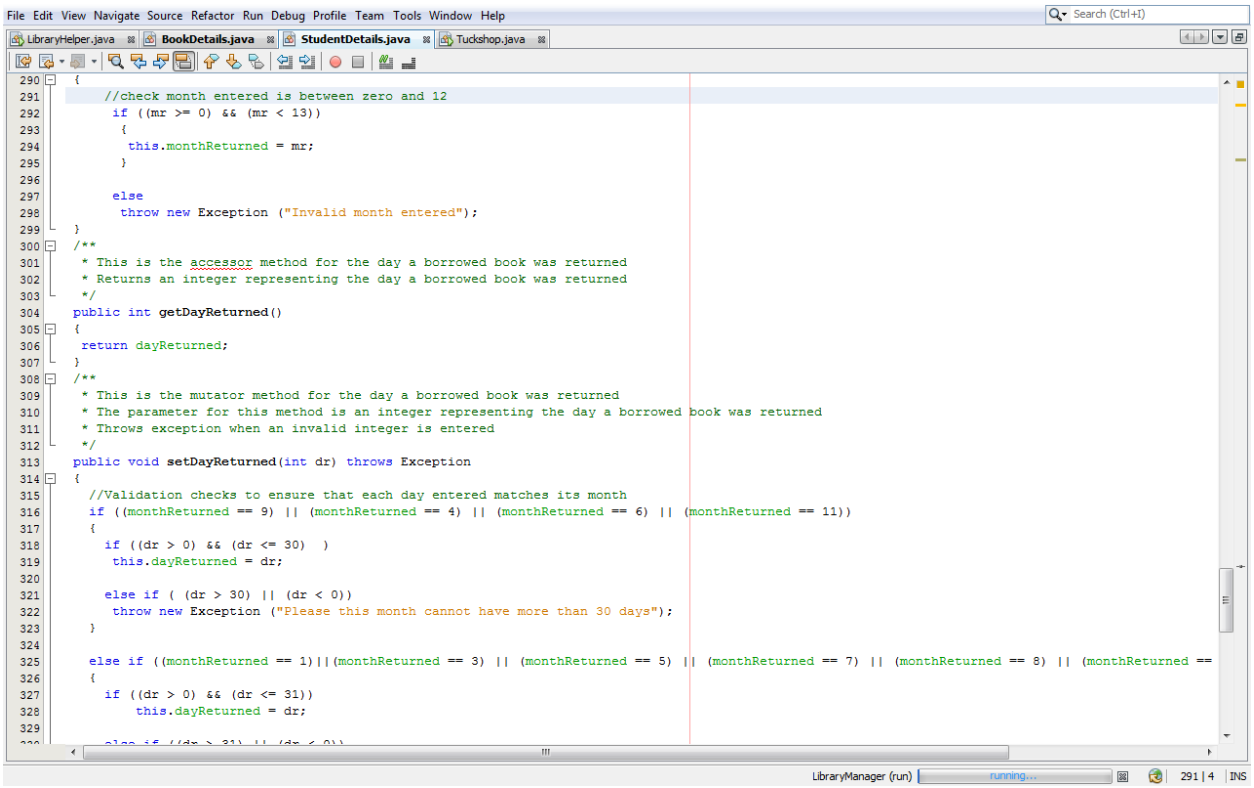
38.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
return grade;
}
/**
* This is the mutator method for the grade of a student
* The parameter for this method is an integer representing the grade of a student
* Throws exception when the invalid grade is entered
*/
public void setGrade(int g) throws Exception
{
//Validation check to ensure that the grade entered is between 7 and 12
if ((g > 6) && (g < 13))
this.grade = g;
else
throw new Exception ("Please grade entered can only be between 7 -12");
}
/**
* This is the accessor method for the Id of the book borrowed by a student
* Returns a string representing the Id of the book borrowed by the student
*/
public String getBookBorrowed()
{
return bookBorrowed;
}
/**
* This is the mutator method for the Id of the book borrowed by a student
* The parameter for this method is a string representing the Id of a book borrowed by a student
* Throws exception when an invalid Id is entered
*/
public void setBookBorrowed(String bb) throws Exception
{
//check whether the Id entered begins with '000_', or is 'null'
if((bb.substring(0, 4).equals("000_")) || (bb.equals("null")))
this.bookBorrowed = bb;
else
throw new Exception("Incorrect input for Book Borrowed");
}
/**
*This is the accessor method for the year a borrowed book was returned
*Returns an integer representing the year a borrowed book was returned
LibraryManager (run)
running...
219 | 1
INS

39.

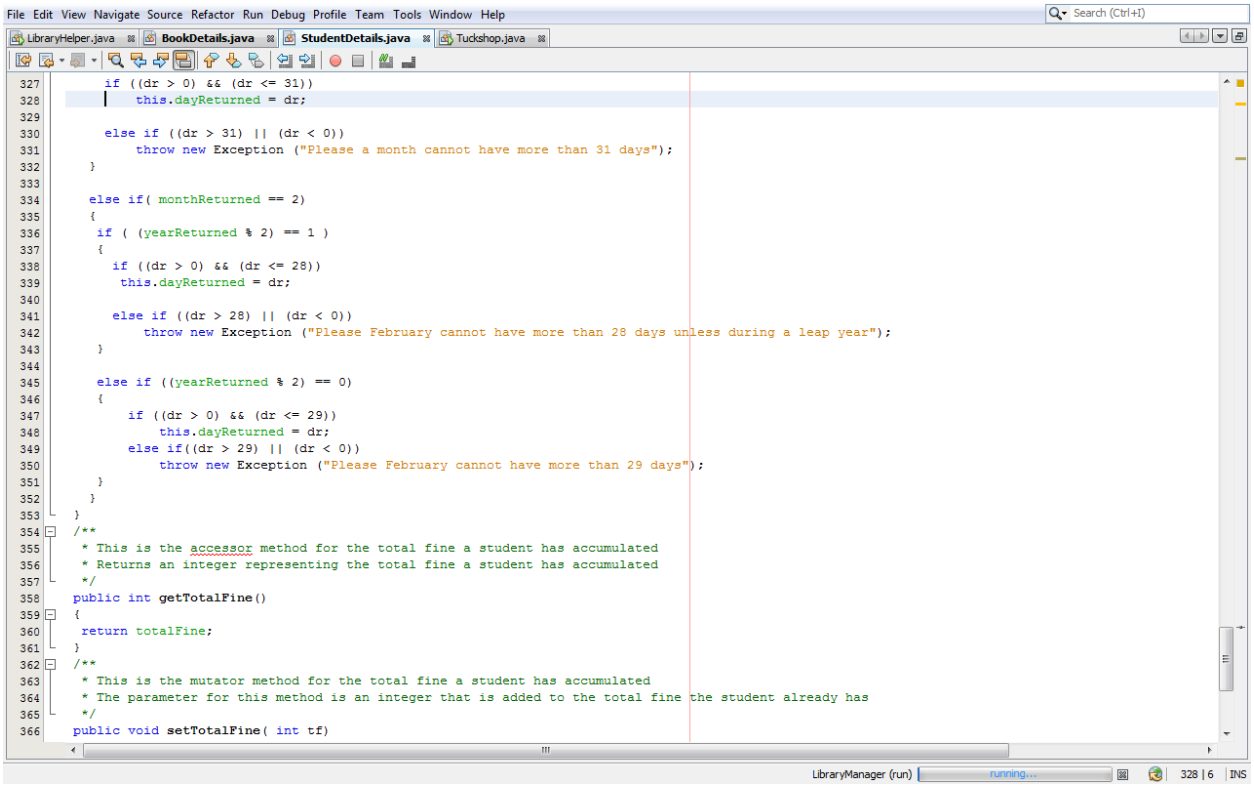

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
LibraryHelper.java
BookDetails.java
StudentDetails.java
Tuckshop.java
}
/**
*This is the accessor method for the year a borrowed book was returned
*Returns an integer representing the year a borrowed book was returned
*/
public int getYearReturned ()
{
return yearReturned;
}
/**
* This the mutator method for the year a borrowed book was returned
* The parameter for this method is an integer representing the year a borrowed book was returned
* Throws exception when an invalid integer is entered
*/
public void setYearReturned(int yr) throws Exception
{
if((yr == 0) || (yr >= 2012))
this.yearReturned = yr;
else if (yr < 0)
throw new Exception("Invalid year entered");
}
/**
* This is the accessor method for the month a borrowed book was returned
* Returns an integer representing the month a borrowed book was returned
*/
public int getMonthReturned ()
{
return monthReturned;
}
/**
* This is the mutator method for the month a borrowed book was returned
* The parameter for this method is an integer representing the month a borrowed book was returned
* Throws exception when an invalid integer is entered
*/
public void setMonthReturned(int mr) throws Exception
{
//check month entered is between zero and 12
if ((mr >= 0) && (mr < 13))
{
LibraryManager (run)
running...
254 | 1
INS

40.



```
290  {
291      //check month entered is between zero and 12
292       if ((mr >= 0) && (mr < 13))
293        {
294         this.monthReturned = mr;
295        }
296
297       else
298        throw new Exception ("Invalid month entered");
299  }
300  /**
301   * This is the accessor method for the day a borrowed book was returned
302   * Returns an integer representing the day a borrowed book was returned
303   */
304  public int getDayReturned()
305  {
306   return dayReturned;
307  }
308  /**
309   * This is the mutator method for the day a borrowed book was returned
310   * The parameter for this method is an integer representing the day a borrowed book was returned
311   * Throws exception when an invalid integer is entered
312   */
313  public void setDayReturned(int dr) throws Exception
314  {
315    //Validation checks to ensure that each day entered matches its month
316    if ((monthReturned == 9) || (monthReturned == 4) || (monthReturned == 6) || (monthReturned == 11))
317    {
318      if ((dr > 0) && (dr <= 30)  )
319       this.dayReturned = dr;
320
321      else if ( (dr > 30) || (dr < 0))
322       throw new Exception ("Please this month cannot have more than 30 days");
323    }
324
325    else if ((monthReturned == 1)||(monthReturned == 3) || (monthReturned == 5) || (monthReturned == 7) || (monthReturned == 8) || (monthReturned ==
326    {
327      if ((dr > 0) && (dr <= 31))
328          this.dayReturned = dr;
329
```

41.



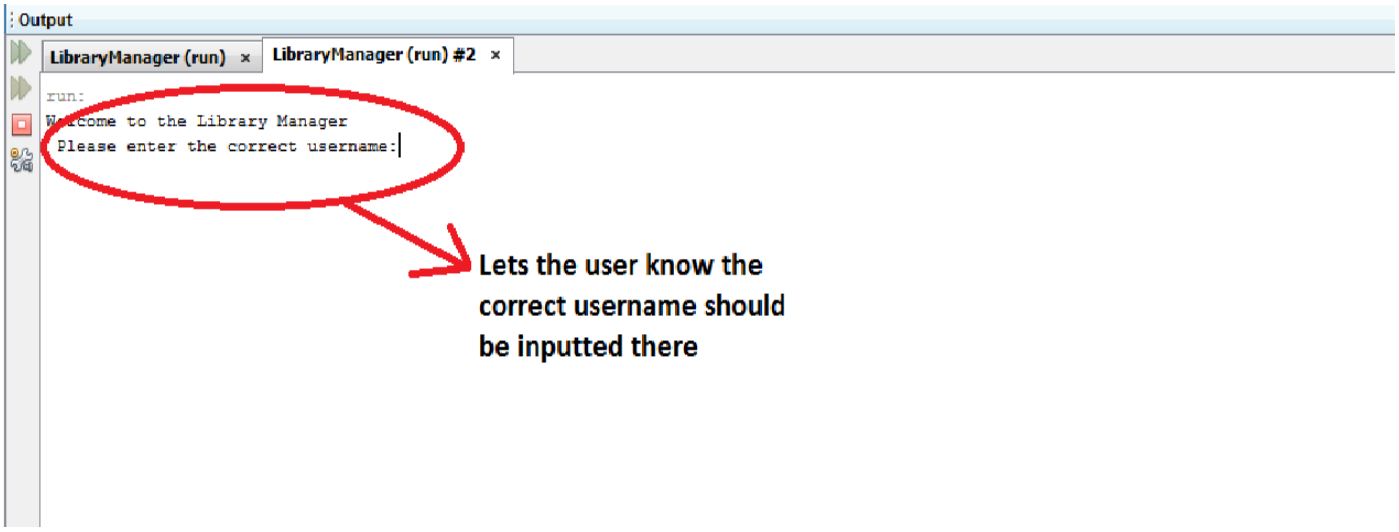
```
327      if ((dr > 0) && (dr <= 31))
328          this.dayReturned = dr;
329
330      else if ((dr > 31) || (dr < 0))
331          throw new Exception ("Please a month cannot have more than 31 days");
332    }
333
334    else if( monthReturned == 2)
335    {
336     if ( (yearReturned % 2) == 1 )
337     {
338       if ((dr > 0) && (dr <= 28))
339        this.dayReturned = dr;
340
341       else if ((dr > 28) || (dr < 0))
342           throw new Exception ("Please February cannot have more than 28 days unless during a leap year");
343     }
344
345     else if ((yearReturned % 2) == 0)
346     {
347         if ((dr > 0) && (dr <= 29))
348             this.dayReturned = dr;
349         else if((dr > 29) || (dr < 0))
350             throw new Exception ("Please February cannot have more than 29 days");
351     }
352    }
353  }
354  /**
355   * This is the accessor method for the total fine a student has accumulated
356   * Returns an integer representing the total fine a student has accumulated
357   */
358  public int getTotalFine()
359  {
360   return totalFine;
361  }
362  /**
363   * This is the mutator method for the total fine a student has accumulated
364   * The parameter for this method is an integer that is added to the total fine the student already has
365   */
366  public void setTotalFine( int tf)
```

```
347            if ((dr > 0) && (dr <= 29))
348                this.dayReturned = dr;
349            else if((dr > 29) || (dr < 0))
350                throw new Exception ("Please February cannot have more than 29 days");
351        }
352       }
353    }
354    /**
355     * This is the accessor method for the total fine a student has accumulated
356     * Returns an integer representing the total fine a student has accumulated
357     */
358    public int getTotalFine()
359    {
360     return totalFine;
361    }
362    /**
363     * This is the mutator method for the total fine a student has accumulated
364     * The parameter for this method is an integer that is added to the total fine the student already has
365     */
366    public void setTotalFine( int tf)
367    {
368     this.totalFine =  this.totalFine + tf; //this updates the total fine a student has by adding an integer (tf) to a student's total fine
369    }
370   }
371
372
373
```

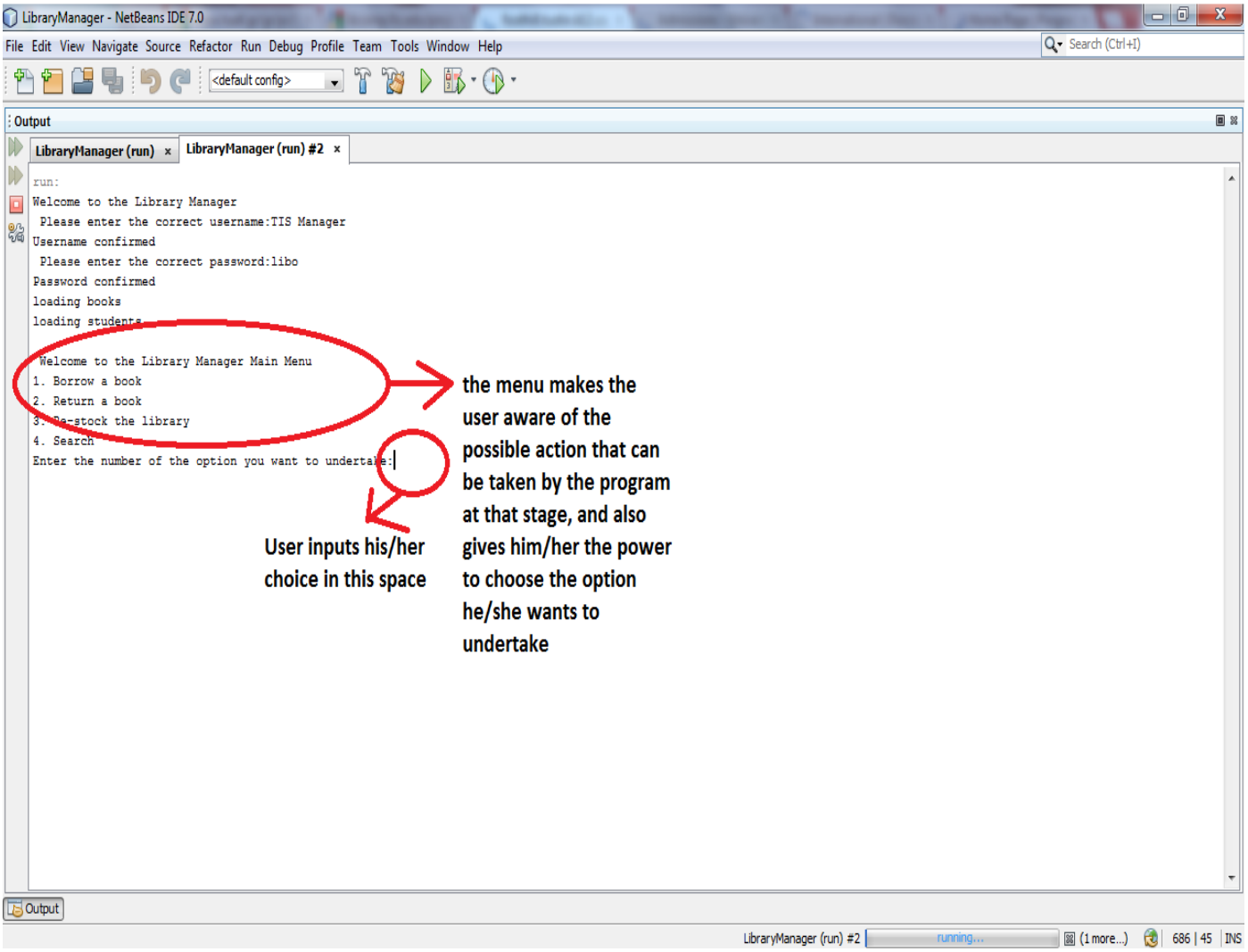
Usability

One of the criteria for the success of this program was for it to have a user-friendly interface that will make the user find the program comfortable to use. In this section, I will discuss the various features of the program that make the program achieve this objective.

One of the most important features of this program that make it user-friendly, is the fact that the program has useful instructions at each stage that allows the user to know what to do during each stage of the program. An example of this feature is shown below:

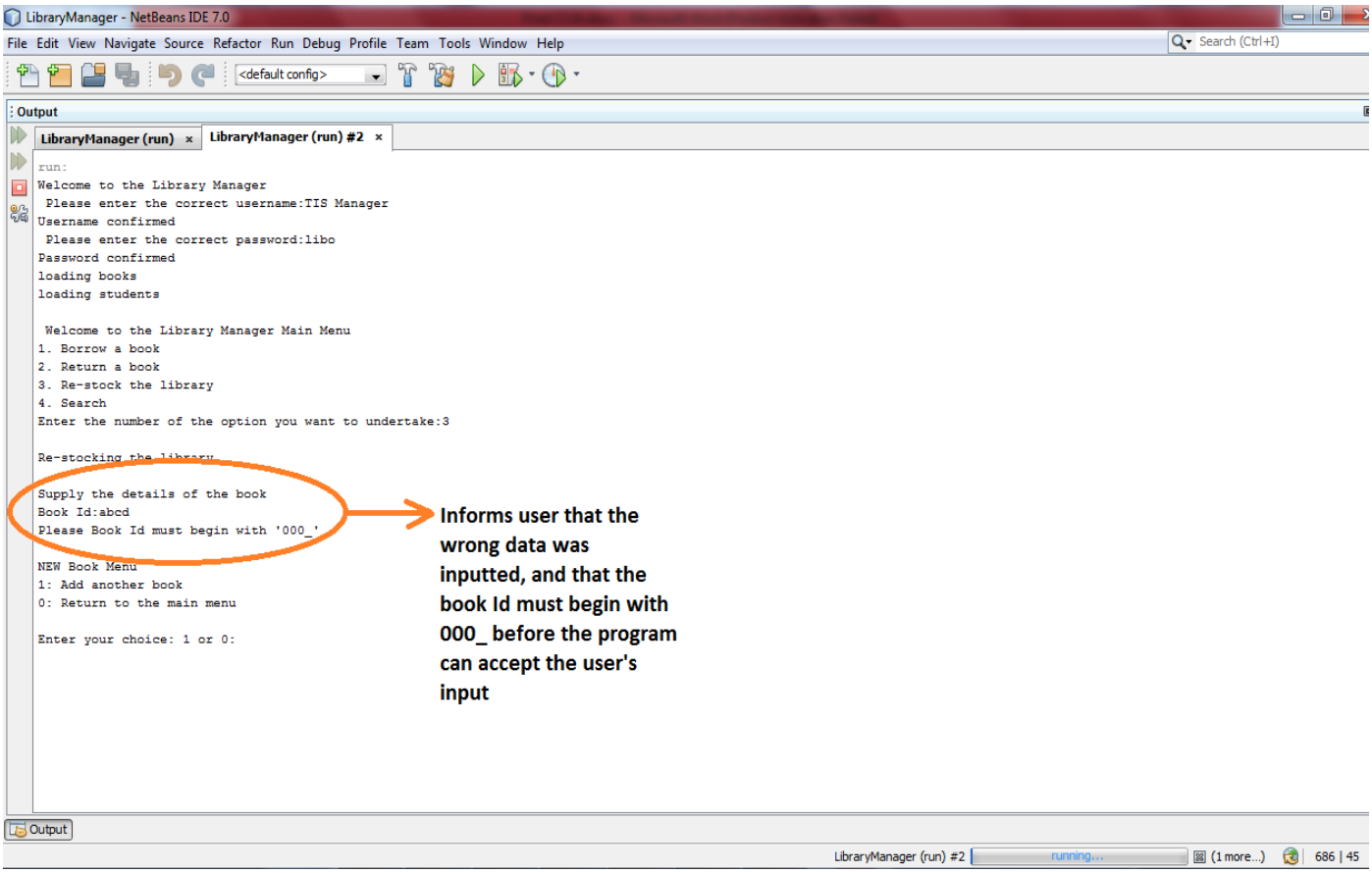


Another feature of this program is the fact that it makes use of helpful menus that give the user options as to the action he/she wants the program to undertake. This feature makes the program more user-friendly by preventing the user from being restricted as to what he/she is allowed to do at certain stages of the program; this feature also makes the user have more control over the actions he/she wants the program to undertake. An example of these menus in use is shown below:



Another feature that also makes this program user-friendly is the fact that it alerts the user when he/she inputs the wrong the data, and then goes on to inform the user about the correct type of input that the program can accept. This prevent the user from continually inputting the wrong type of data, and this

also make the user aware of the correct type of input that is acceptable at each stage of the program. An example of this feature is shown below:



Handling of Errors

As the user is likely to input the wrong type of data into the program at different stages whilst the program is running, the program has been designed to be robust enough to handle anticipated wrong data the user is likely to input into the program. The following section will document the various ways the program will handle errors that are likely to occur whilst the program is being run.

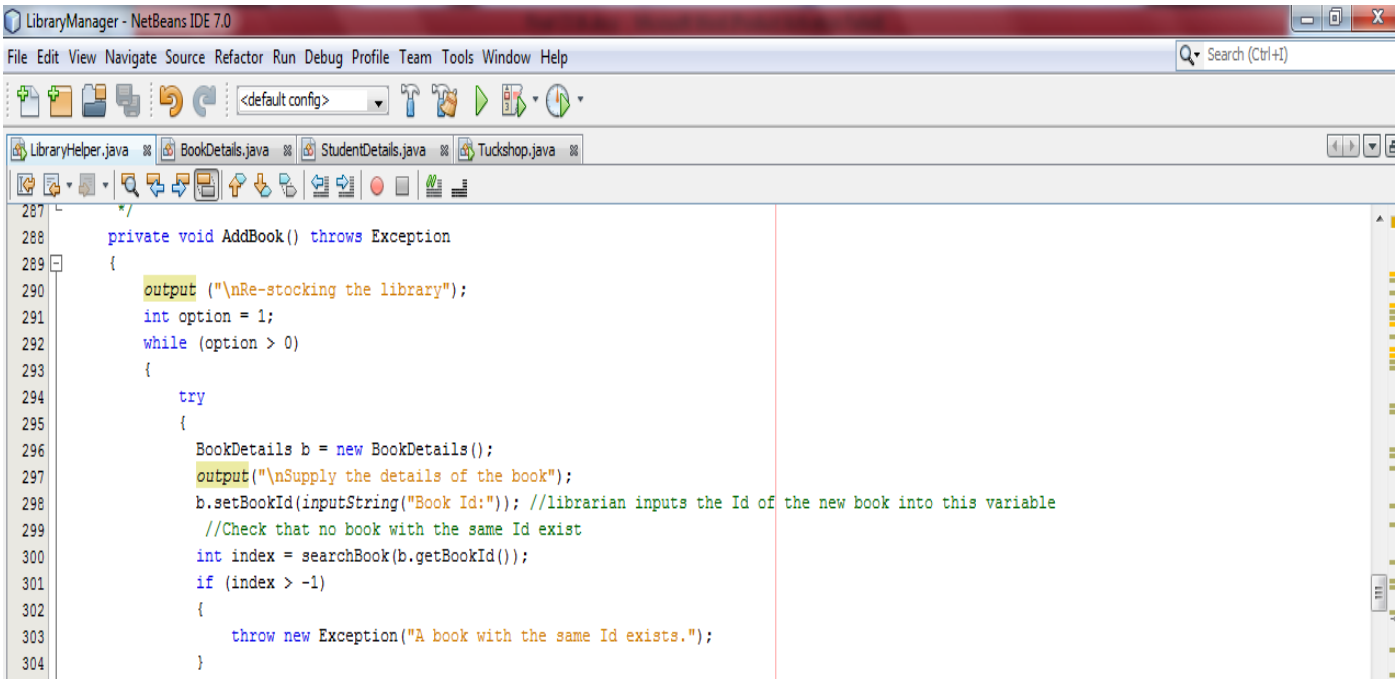
Since the program begins by asking the user to input the correct username and password into the program, the first error the program handles are the wrong data inputs for the password and username. The program runs a ‘while-loop’ during the stage the user is supposed to input the correct password and username, so that the program keeps asking the user to input the correct password and username until the user inputs the correct username and password. The code for the handling of the incorrect username is shown below:

```
32      public LibraryHelper() throws Exception
33      {
34          output ("Welcome to the Library Manager");
35          String correctUsername = "TIS Manager"; // sets the valid username for the program to run
36          String correctPassword = "libo"; // sets the valid password for the program to run
37          String inputPassword; // variable that stores the password the user inputs into the program
38          String inputUsername; // variable that stores the username the user inputs into the program
39
40          //validation check toensure that the inputted username matches the valid username required for the program to run
41          int usernameValidation; //this is the variable that will compare the inputted username to the correct username
42          do
43          {
44          inputUsername = inputString (" Please enter the correct username:");
45          usernameValidation = correctUsername.compareTo(inputUsername); //compares the inputted username to the correct username
46          } while (usernameValidation != 0);
47          output("Username confirmed");
48
```

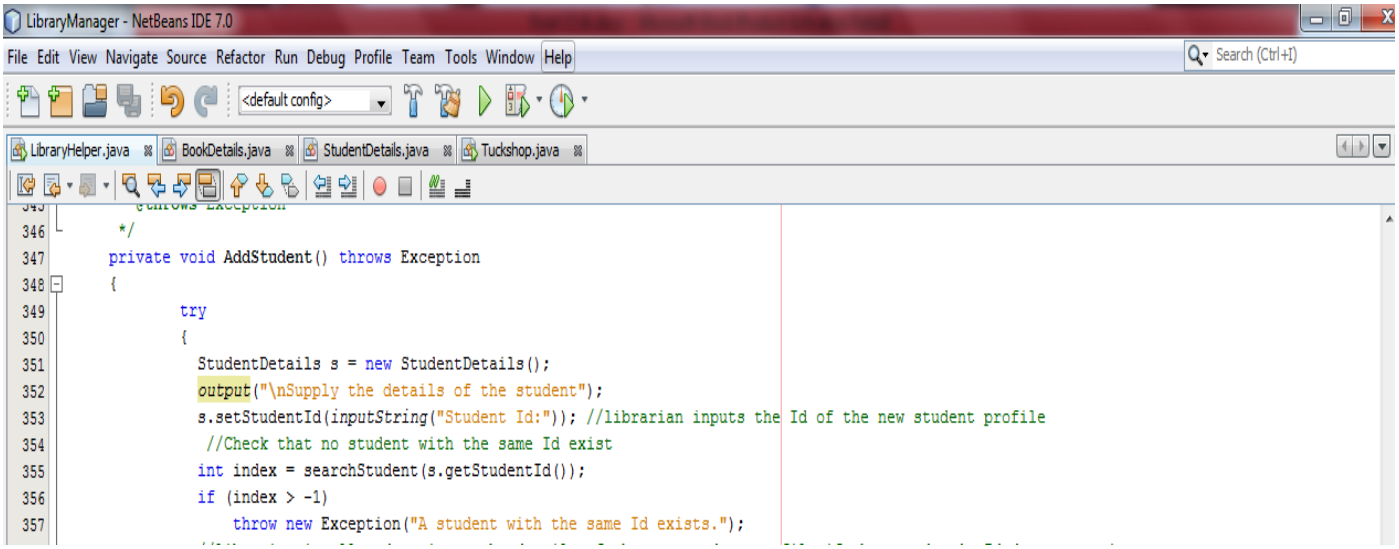
And the code for handling the incorrect password is shown below:

```
48
49          //validation check to ensure that the inputted password matches the valid password required for the program to run
50          int passwordValidation;
51          do
52          {
53           inputPassword = inputString (" Please enter the correct password:");
54           passwordValidation = correctPassword.compareTo(inputPassword); //comparres the inputted password to the correct password
55          } while (passwordValidation != 0);
56          output("Password confirmed");
57
```

During the stage a book profile is being added to the library database, the has to input the Id of the new book being added to the library database; at this stage, the user has to enter an Id that does not exist in the library database since the book being added to library database is new, and thus its book Id should also be a new one. Therefore, during this stage the program displays an error message if the user enters a book Id that exists in the library database. The program does this by searching through the bookDetails array list, and if the program finds the book Id entered, then the program throws an error to show that the book Id entered already exists. The code for this is shown below:



During the stage a new student profile is being added to the library database, the librarian also has to input the Id of the new student profile being added. Similar to the stage when a new book profile was being added, the program also ensures that the student Id entered by the user does not exist before the librarian can input the other details of the new student profile. Thus, when the user inputs a student profile that already exists, the program throws an error that displays to the user that the student Id entered already exists. The code for this is shown below:



During the stage a book is being borrowed from the library, the librarian has to first input the Id of the book being borrowed; at this stage, the Id entered into the program must exist in the library database, since only books that exist in the library database can be borrowed. Thus during this stage the program looks through the bookDetails array list to check whether the inputted book Id exists or not, if the inputted book Id does not exist, the program then displays an error message to show that the inputted book Id does not exist. The code for this is shown below:

```
469         //throw an exception if the book Id does not exist
470         else if(index <= -1)
471           throw new Exception ("Book Id entered does not match the Id of any book in the library database");
```

Also, during the borrowing stage, a book can only be borrowed if the book is in the library, and is not currently being borrowed; thus, during this stage, the program checks whether the status of the inputted book Id is 'available' or 'unavailable'. If the status of the book is 'unavailable', then the book is currently being borrowed and thus cannot be borrowed at that point in time, thus the program displays an error message to the user to show that the book cannot be borrowed since it is currently being borrowed. The code for this is shown below:

```
463         //throw an exception if the book has already been borrowed
464         else if (books.get(index).getStatus().equalsIgnoreCase("unavailable"))
465         {
466          throw new Exception ("This book has already been borrowed");
467         }
```

A student can only one book at a time, thus during the borrowing stage, if the student borrowing the book is found to already have a borrowed book, the program displays an error message to show the librarian that the student already has a book out. The program knows that the student already has a book out, if the student's profile has a book Id in the bookBorrowed section. The code for handling this error is shown below:

```
449             //throws exception if the student currently has another borrowed book in his/her possession
450            else if(student.get(studentIndex).getBookBorrowed().length() > 4)
451             {
452              throw new Exception ("A student cannot borrow more than one book; this student has already borrowed a book");
453             }
```

When a book is being borrowed from the library, the librarian has to input the Id of the book being borrowed; at this stage, the Id inputted by the librarian has to exist in the library database since it is only books in the library database that can be returned to the library. Thus, during this stage the program searches through the bookDetails array list for the inputted book Id, and if the book Id is not found, the program displays an error message to show that the book Id entered does not exist in the library database. The code for handling this error is shown below:

```
567             //throw exception if the book Id does not exist
568             else if (index <= -1)
569                throw new Exception("Book Id not found");
```

A book can only be returned, if it was borrowed in the first place, thus during the stage a book is being returned the program check whether the status of the book reads 'unavailable', and if the status reads 'unavailable' then it means that the book was out of the library and thus it was borrowed; this implies that that book can be returned. However, if the status of the book reads 'available', then it implies that the book was never borrowed, and thus cannot be returned to the library, and the program makes this error known to the librarian by displaying an error message to show this. The code for handling this error is shown below:

```
563             //throw exception if the book was never borrowed
564             else if(books.get(index).getStatus().equalsIgnoreCase("available"))
565               throw new Exception(" Please this book has not been borrowed");
```

After the Id of the book being returned is entered, the user has to input the Id of the student returning the book; at this stage the Id of the student must exist in the library database since only students whose profiles exist in the library database are allowed to borrow book from the library, so only students whose profiles exist in the library database are allowed to return books. Thus, during this stage, the program searches through the studentDetails array list for the inputted student Id, and if the student Id does is not found, the program displays an error message to show this. The code for this is shown below:

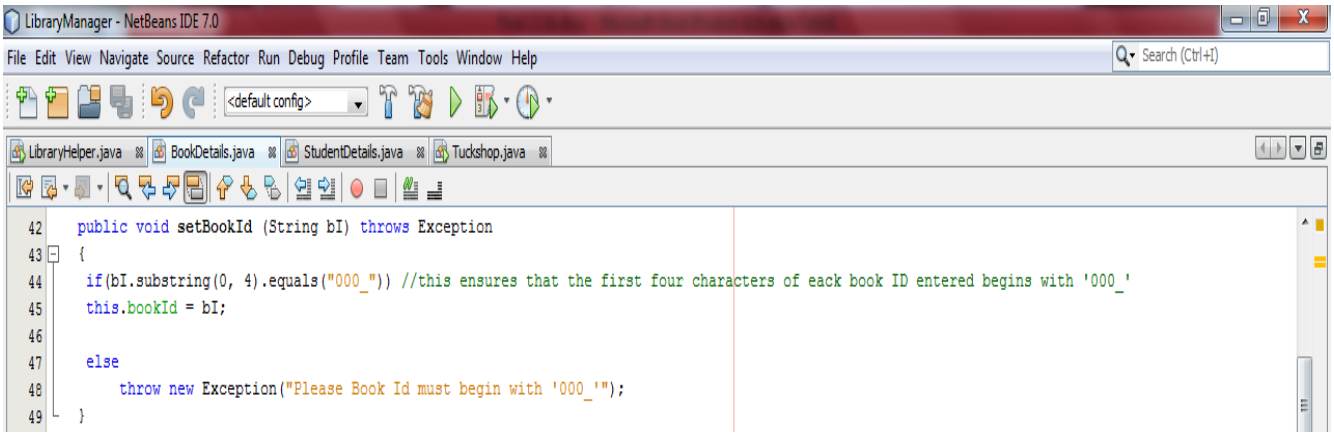
```
559                 //throw exception if the student Id does not exist
560                 else if (studentIndex <= -1)
561                     throw new Exception ("Please the student Id entered does not match the Id of any student in the library database");
```

Now after the student Id is validated to exist in the library database, the program must check whether the student borrowed the book being returned; this is because a student cannot return a book he/she did not borrow. The program does this by checking whether the book Id in the bookBorrowed section of the student's profile is the same as the book Id inputted by the librarian at the beginning of the stage; if the Ids do not match, an error message is displayed. The code for this is shown below:

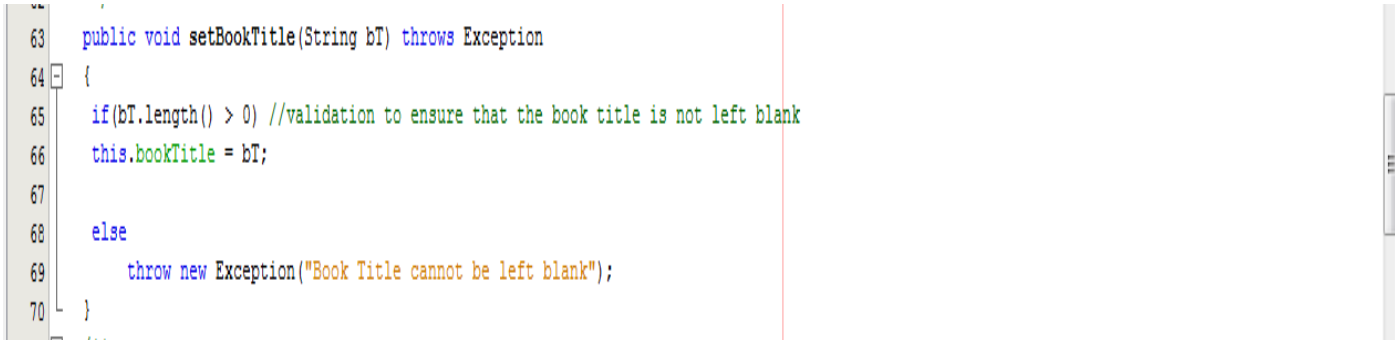
```
553                 //throw exception if the student did not borrow the book being returned
554                 else if(student.get(studentIndex).getBookBorrowed().compareToIgnoreCase(books.get(index).getBookId()) < 0)
555                 {
556                  throw new Exception ("This student has not borrowed the book entered");
557                 }
```

The following errors to be discussed represent the errors that can occur when the profile of a book is being created.

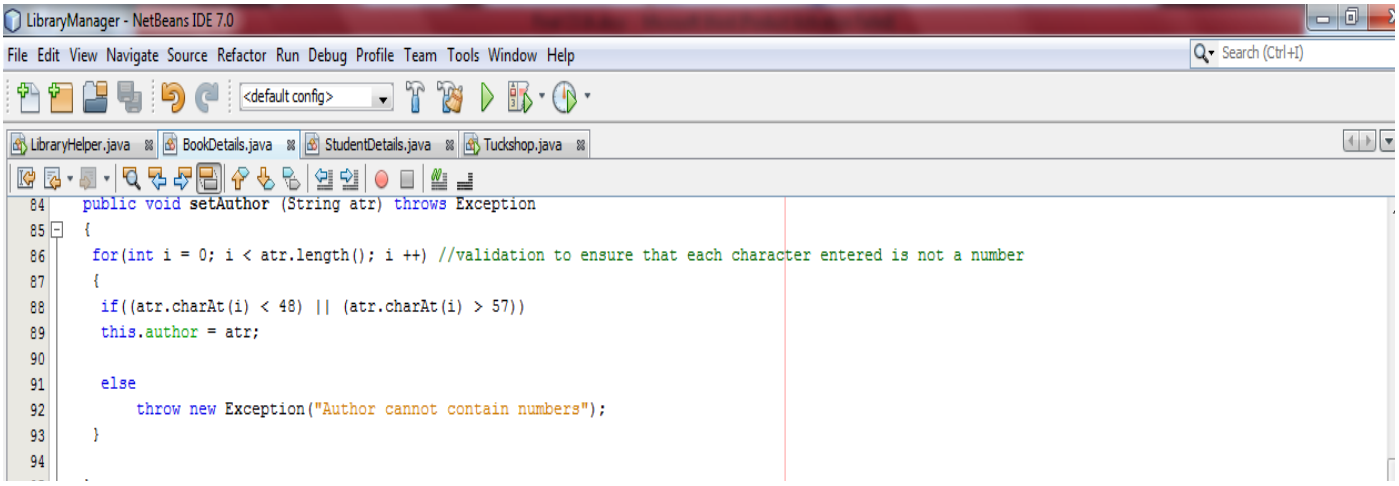
The first of these errors relate to the Id of the book profile being created; since all book Id must begin with the sequence '000_', the program throws an error when the book Id entered by the user does not begin with this sequence. The code for this is shown below:



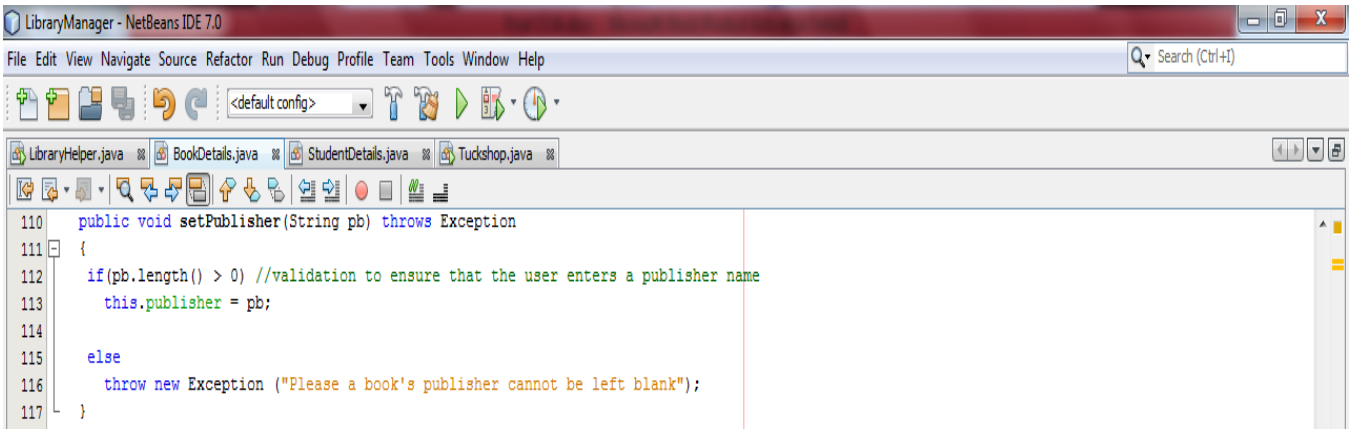
The next relates to the title of a book; a book title cannot be left blank, so the program throws an error when the user leaves the book title blank. The code for this is shown below:



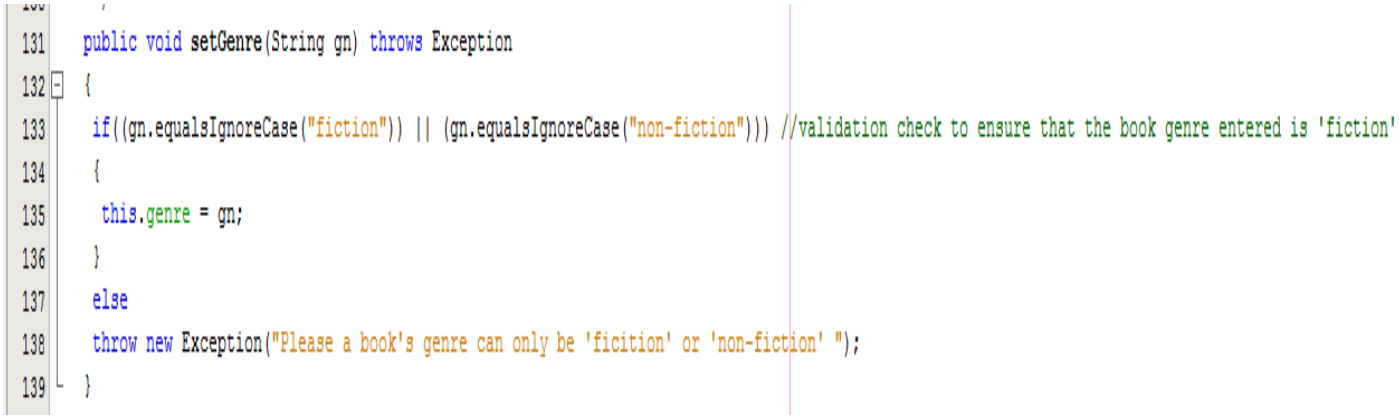
The name of the author of a book cannot contain numbers, thus the program throws an error when the user inputs numbers when the name of the author of a book is being asked for. The code for handling this error is shown below:



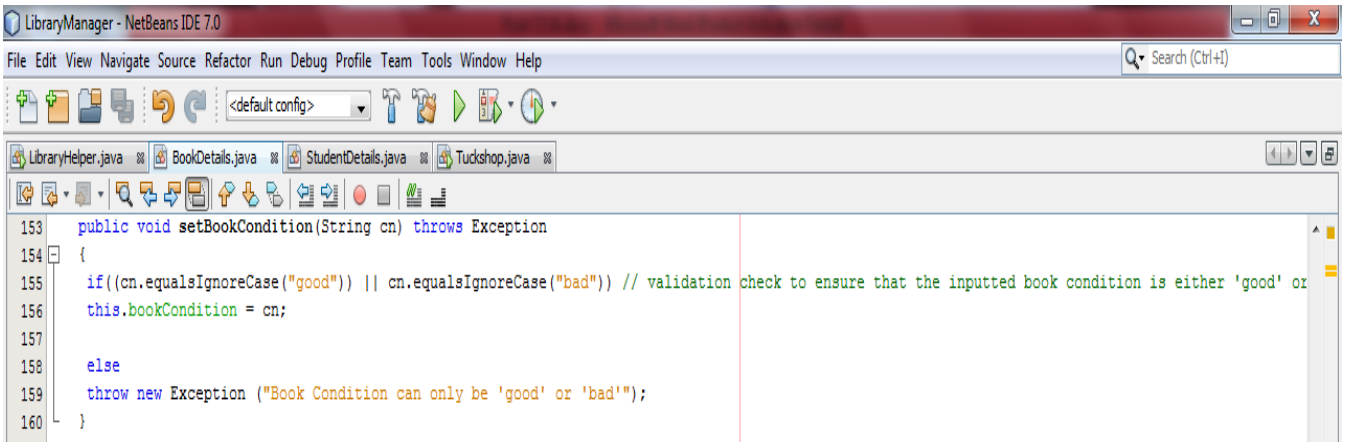
The name of the publisher of a book cannot be left blank when it is being asked for, thus the program throws an error when the user leaves this blank. The code for handling this error is shown below:



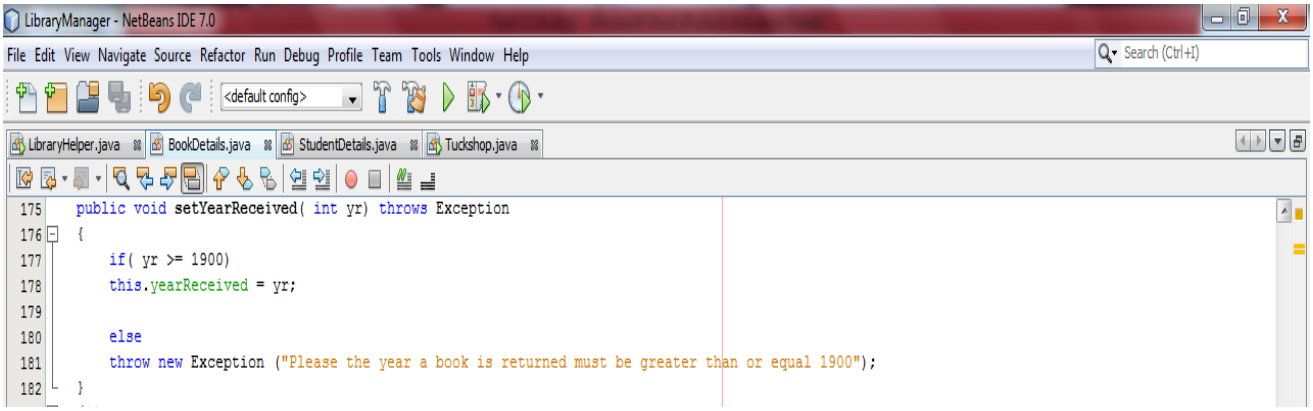
The genre of a book can only be ‘fiction’ or ‘non-fiction’, thus an error occurs when anything other than this is entered by the user; when this happens, an error is displayed to show this. The code for this is shown below:



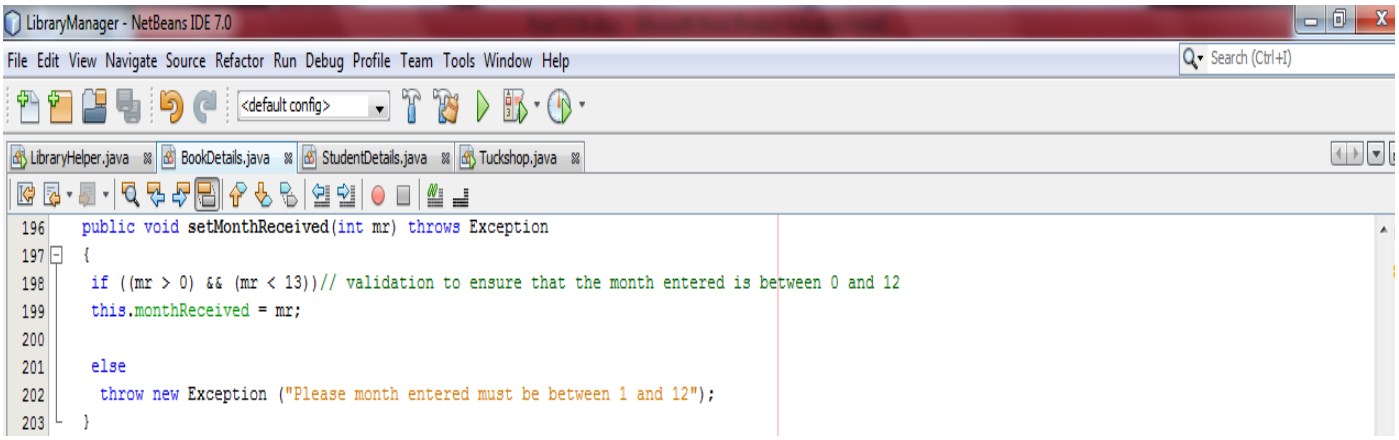
The condition of a book can only be ‘good’ or ‘bad’, thus an error message is displayed when anything other than this is entered by the user. The code for handling this error is shown below:



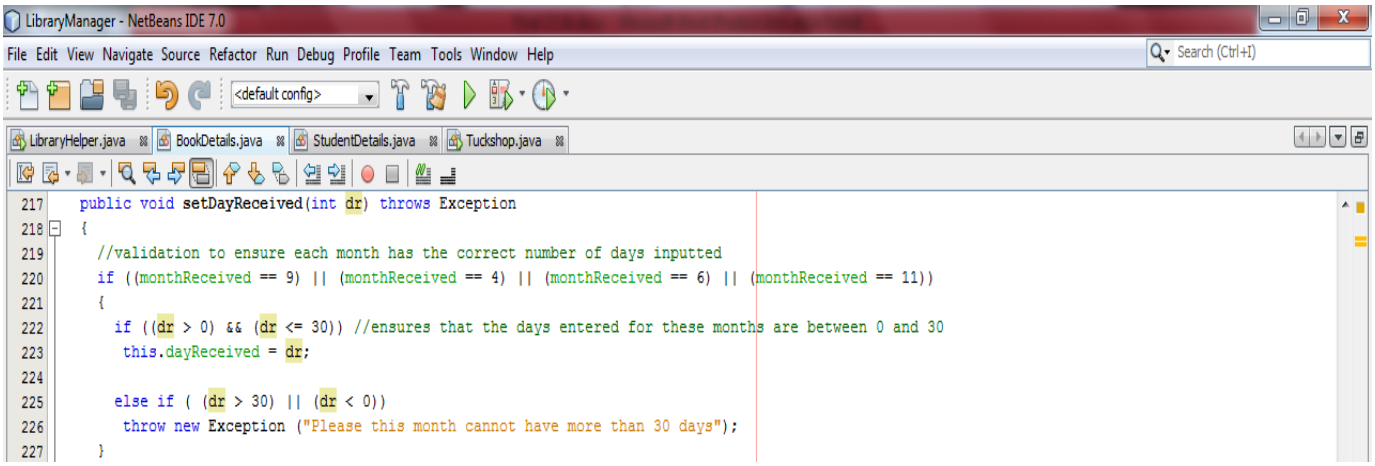
The time frame given to the year a book was stocked into the library was from 1900, this was to give the user a large enough time frame to input the year a book was stocked into the library; thus any year before 1900 becomes an error for the program, and thus it displays an error message when a year before 1900 is inputted by the user. The code for handling this error is shown below:



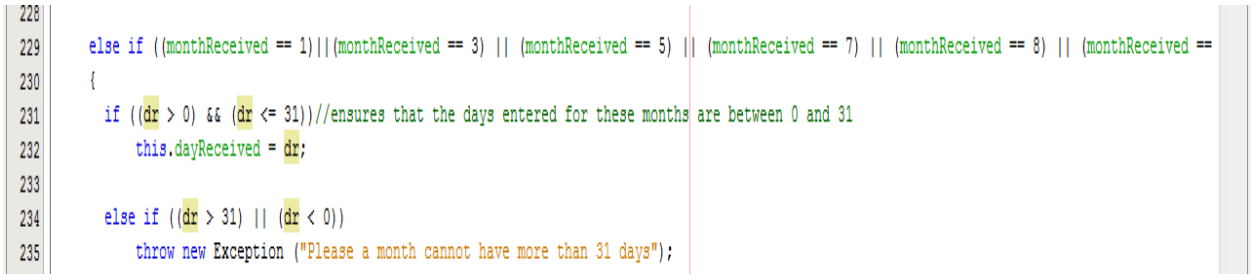
A month can only be between 1 and 12; thus when the user is inputting the month a book was stocked into the library, the program throws an exception if this input is out of the range for a valid month. The code for this is shown below:

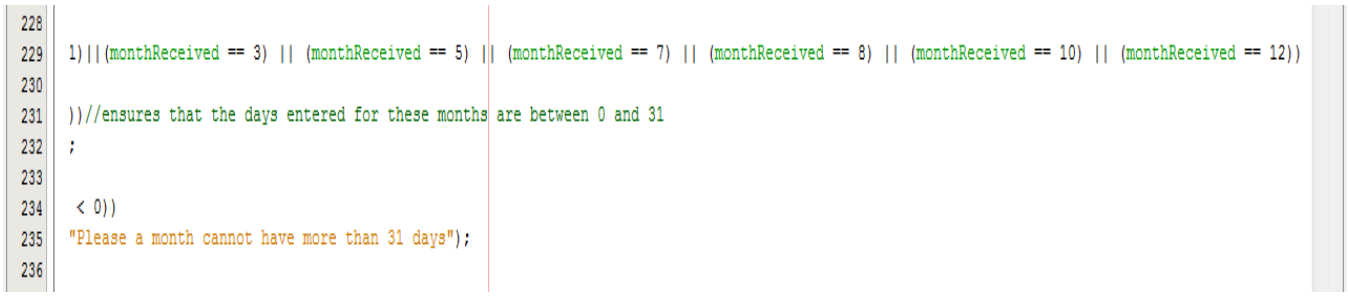


The months of September, April, June and November can only have a maximum of 30 days, thus the program throws an error the number of days for these months are zero, negative, or above 30. The code for handling this error is shown below:

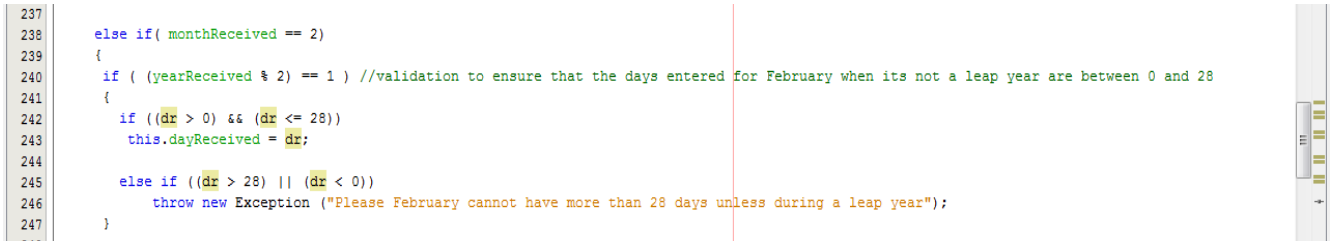


However, the other months except February, can have a maximum of 31 days, thus any input that is zero, negative, or greater than 31 becomes an erroneous input; and, thus the program displays an error message when this error is detected. The code for this is shown in the diagrams below:

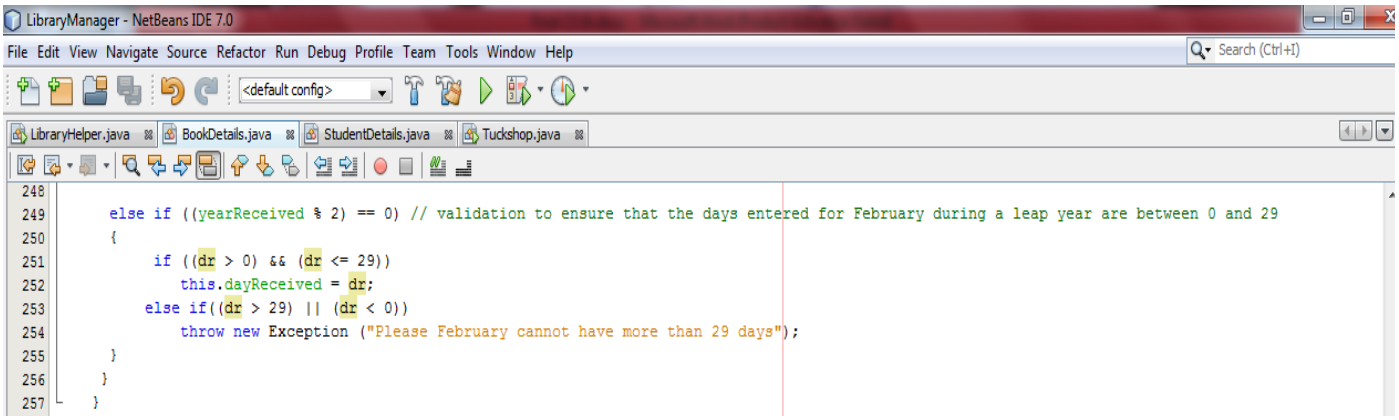




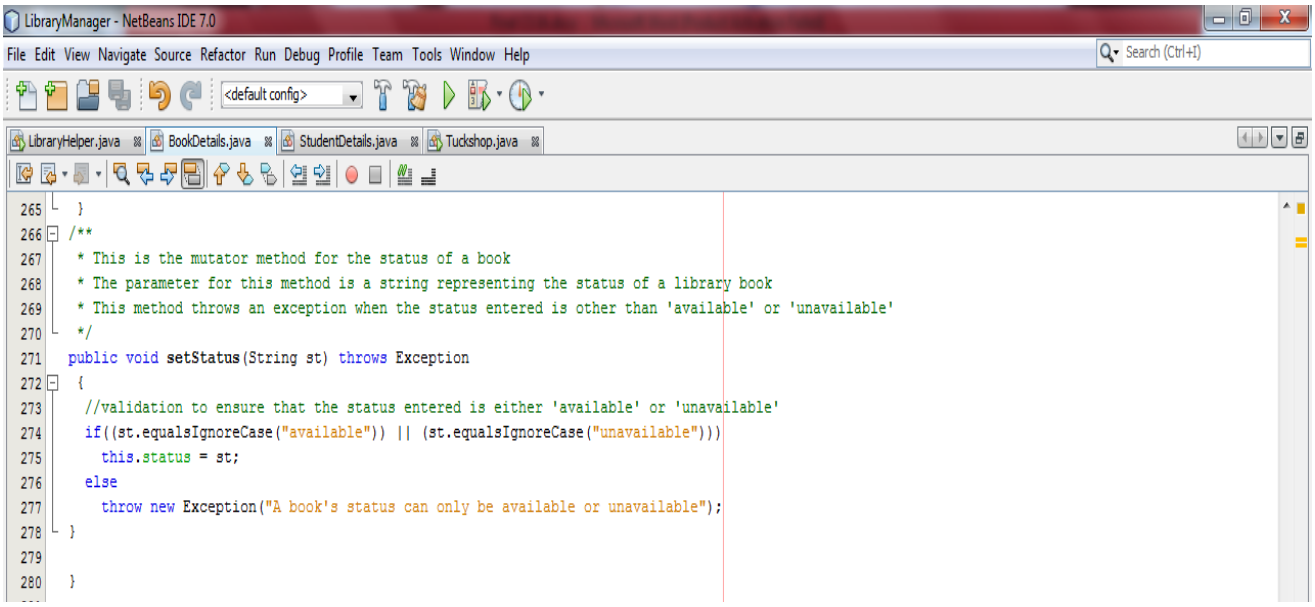
Since February can only have 28 days when it's not a leap year, the program displays an error when a negative input, zero, or a number greater than 28 is inputted for February when it's not a leap year. The code for this is shown below:



However, during a leap year February can have a maximum of 29 days; thus any input that is greater than 29, zero, or negative becomes an error, and the program then displays an error message. The code for this is shown below:

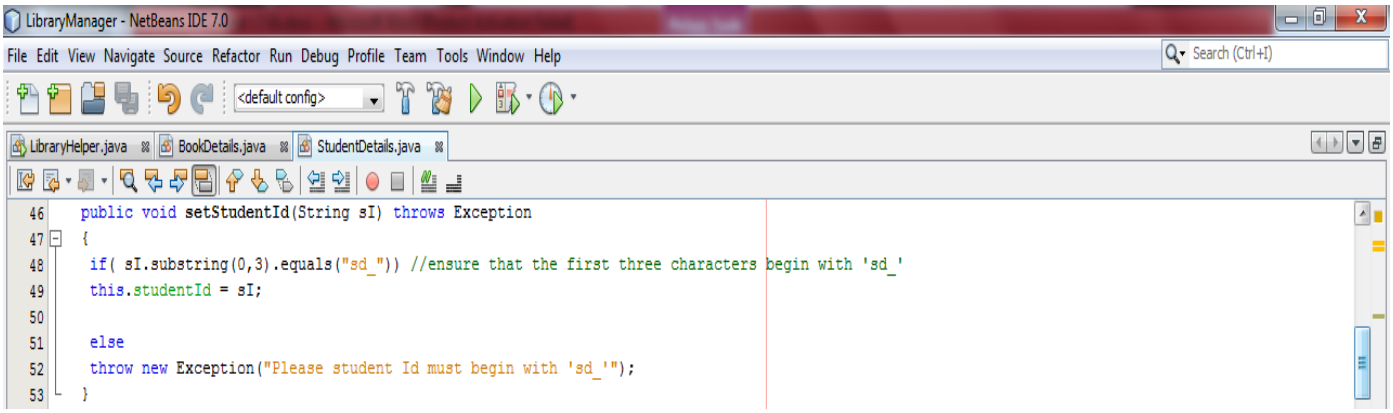


The status of a book can either be 'available' or 'unavailable'; thus any input other than this for the status a book causes the program to throw an exception and display an error message. The code for handling this error is shown below:

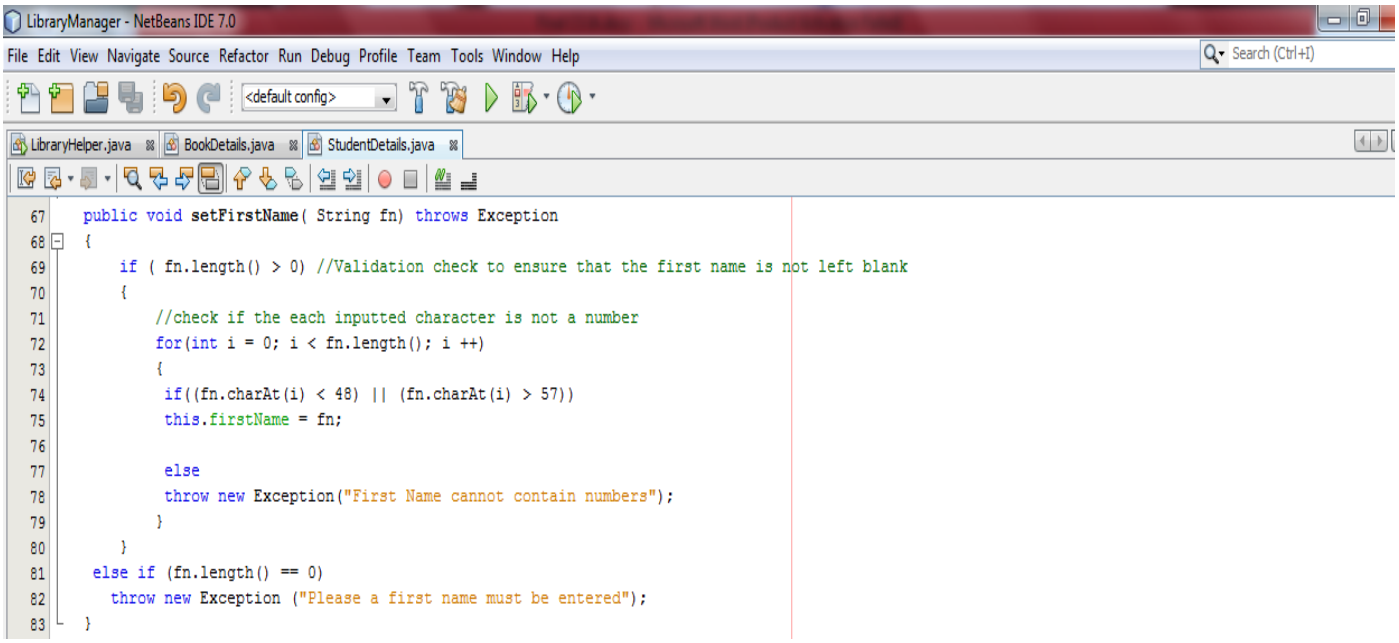


The following errors to be discussed relate to data being inputted for a student's profile.

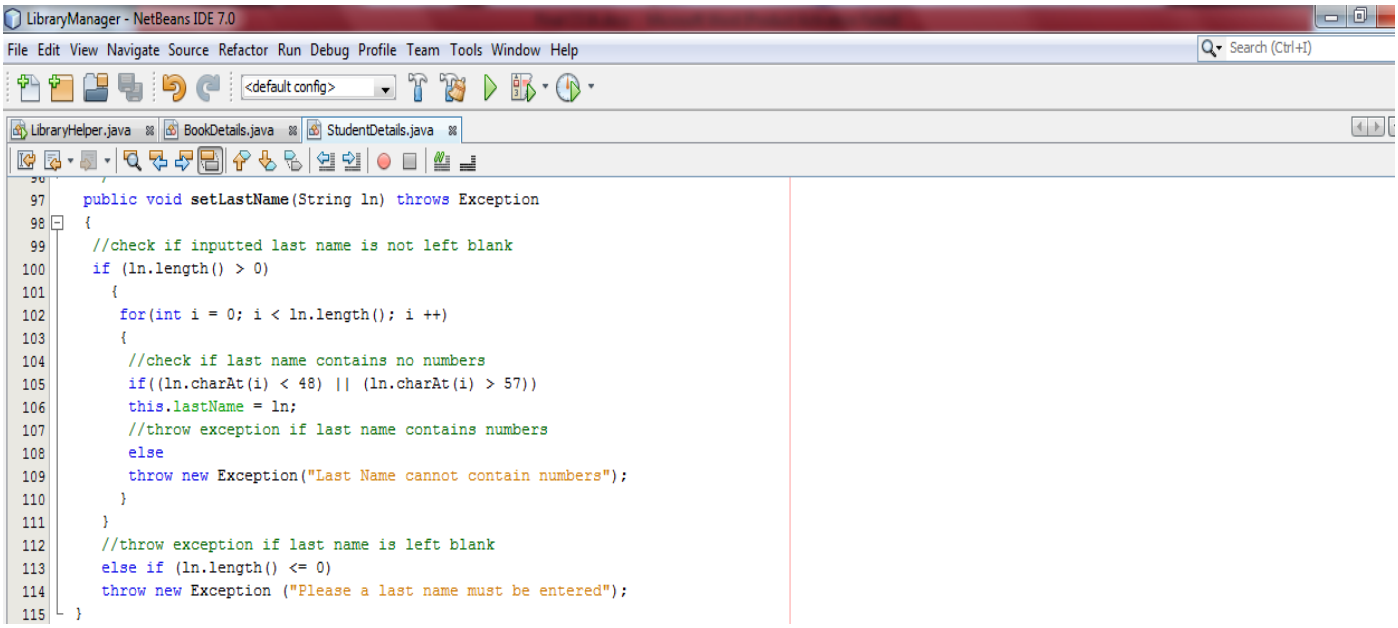
A student Id must begin with 'sd_'; thus when user inputs a student Id that does not begin with this, the program displays an error message. The code for this is shown below:



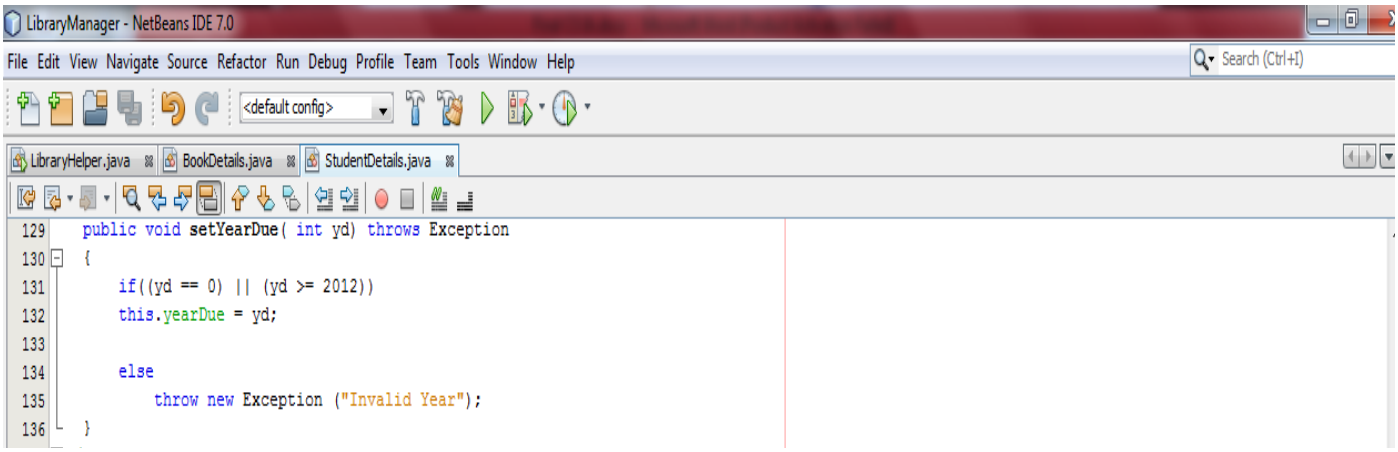
A student first and last cannot be left blank, and cannot also contain numbers; thus an error message is displayed when the user does this. The code for handling this error in the first name is shown below:



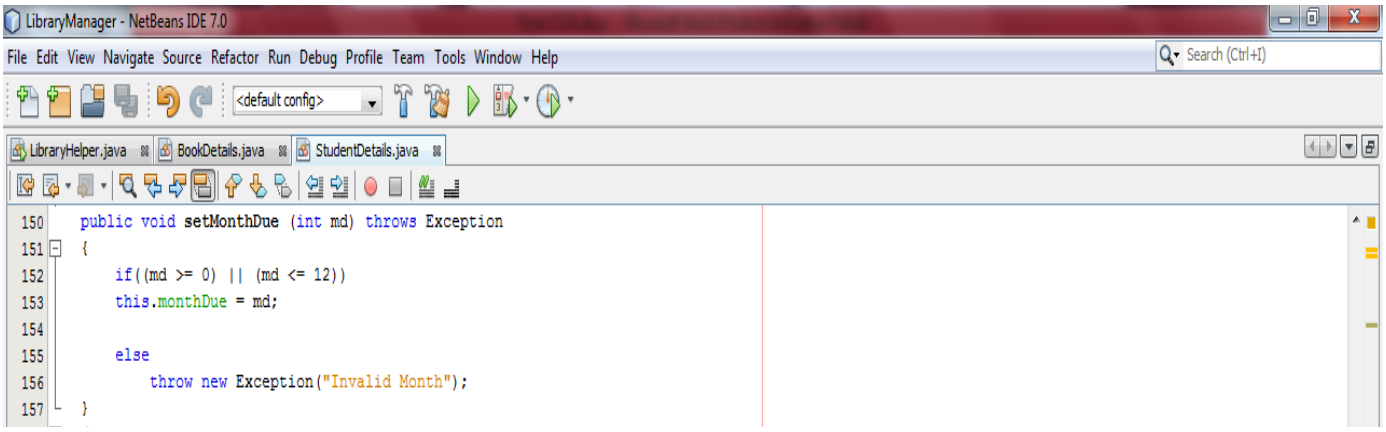
The code for handling this error in the last name is shown below:



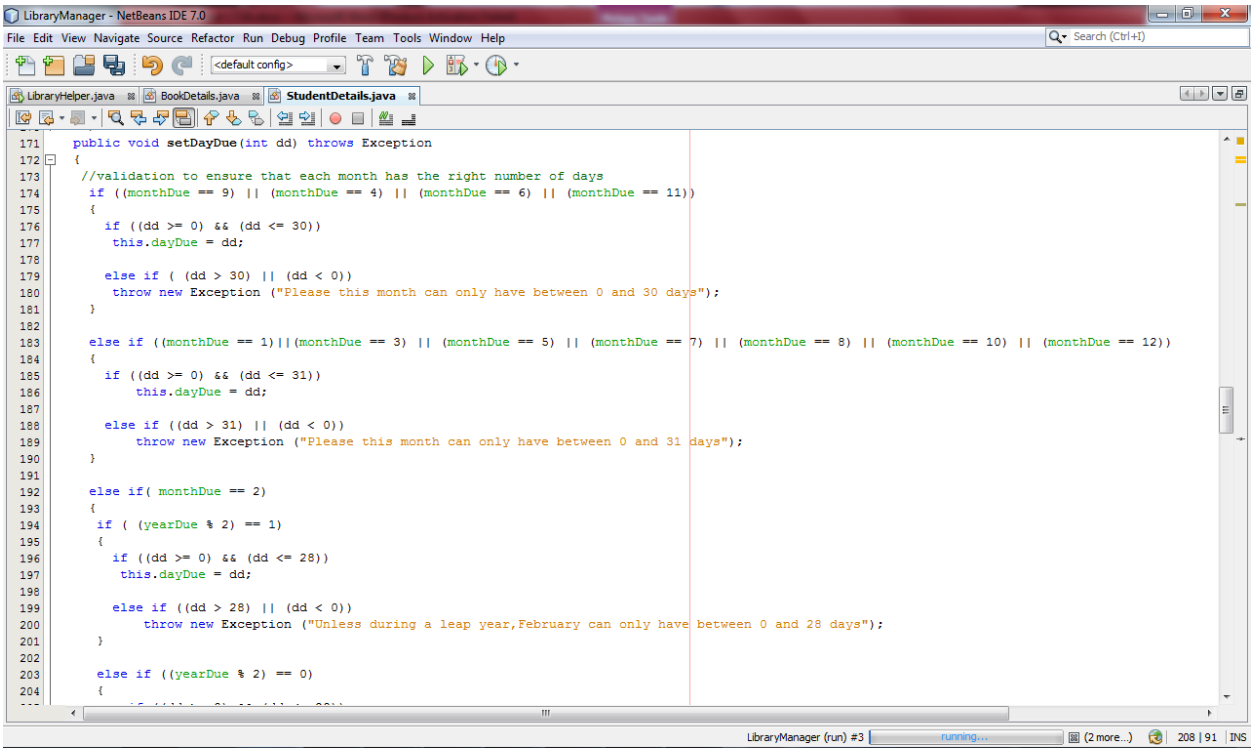
Since the program will be used from 2012 and beyond, and since the year a book is due is set to zero when a book is returned; the acceptable input for the year a book is due is any input that is equal to zero or greater than 2012. Any other input from this causes the program to display an error message. The code for handling this error is shown below:

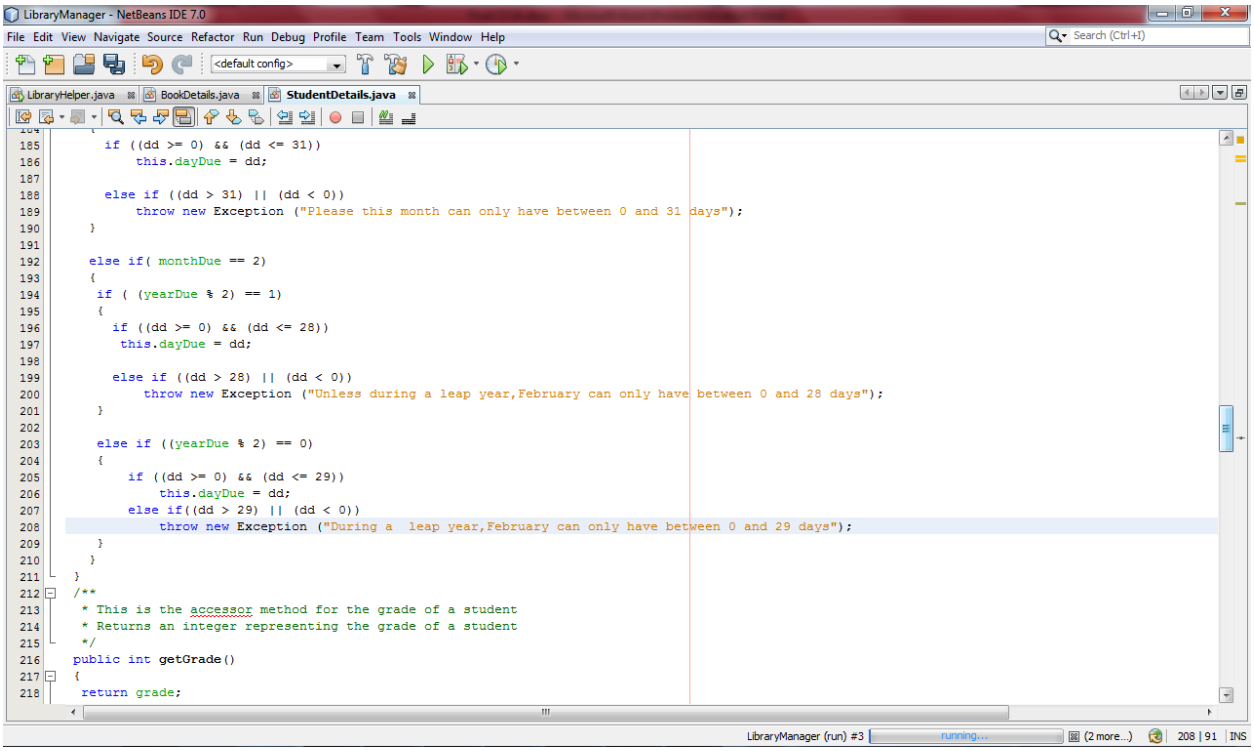


Just like the year a borrowed book is due, the month a borrowed book is due can also be set to zero- this happens during the stage a book is being returned. Thus, the correct inputs for this field is an number from zero to 12; any input apart from this, will be received as erroneous by the program. The code for handling this error is shown below:

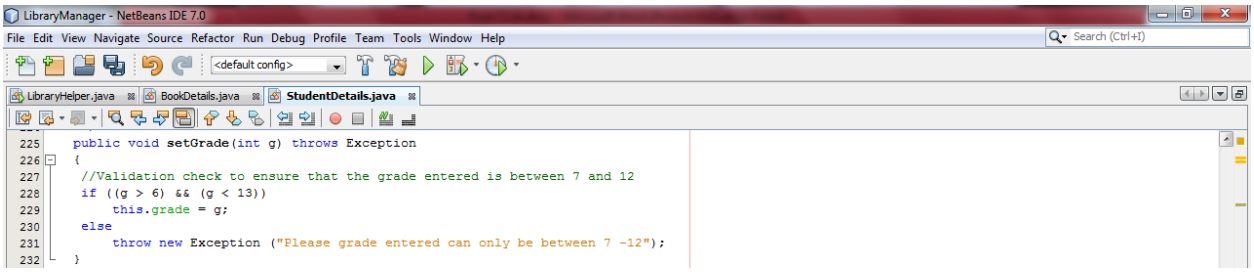


Just like the inputs for the day a book was stocked into the library, the correct inputs for the day a book was returned are the same; the only difference this is the addition of zero as part of the correct inputs. Thus, any other input by the user cause the program to display an error message. The code for handling errors for corresponding months are shown below:

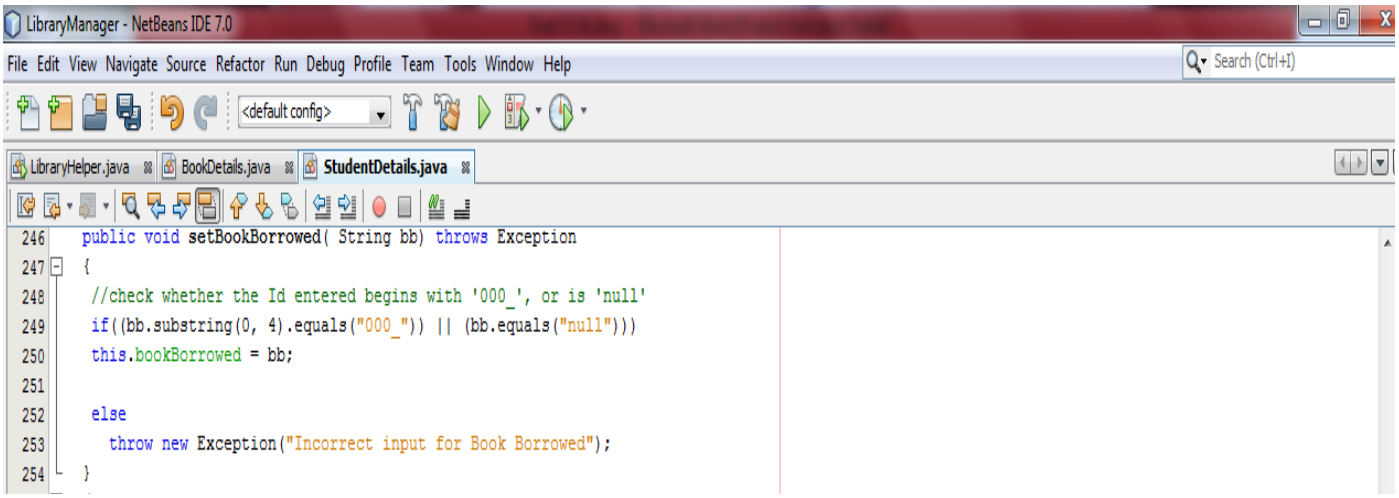




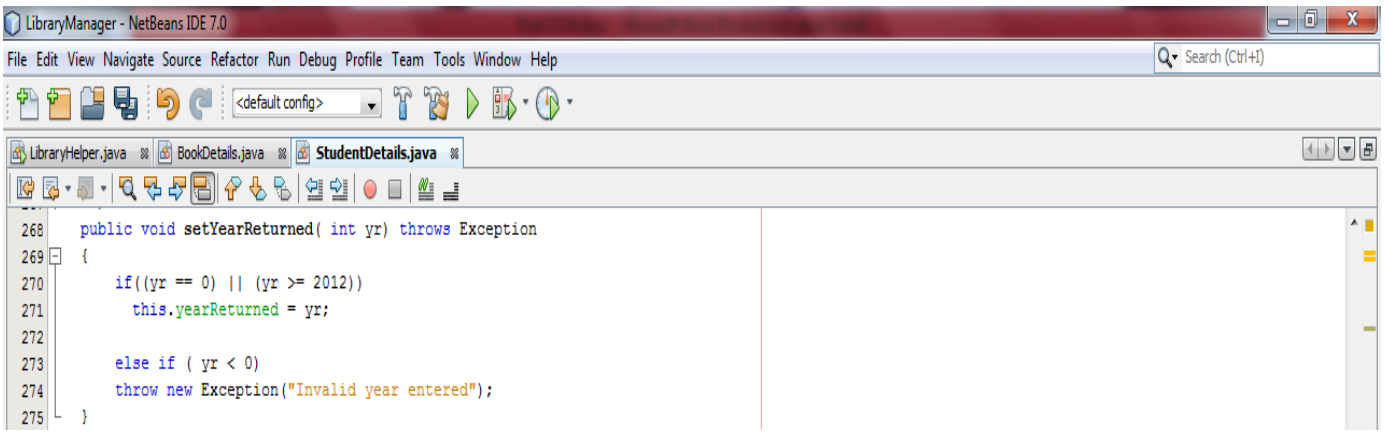
Students in the school are between grades 7 to 12, thus any input other than this would be erroneous. The code for handling this error is shown below:



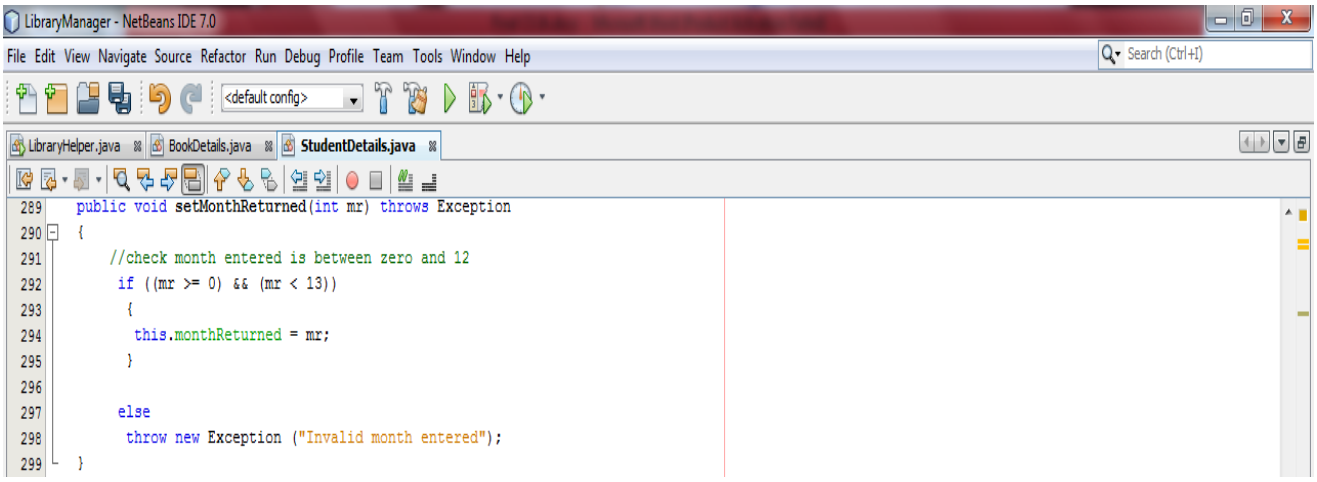
The bookBorrowed field of a student profile stores the Id of a book borrowed by a student, and since all book Id must begin with '000_', data inputted in this field must begin with '000_'; also when a student has not borrowed any book, then this field reads 'null' to show that no book has been borrowed by the student. Thus the correct inputs for this field are any inputs that begin with'000_', or 'null'; any input other than these will be erroneous, and this will cause the program to display an error message. The code for handling this error is shown below:



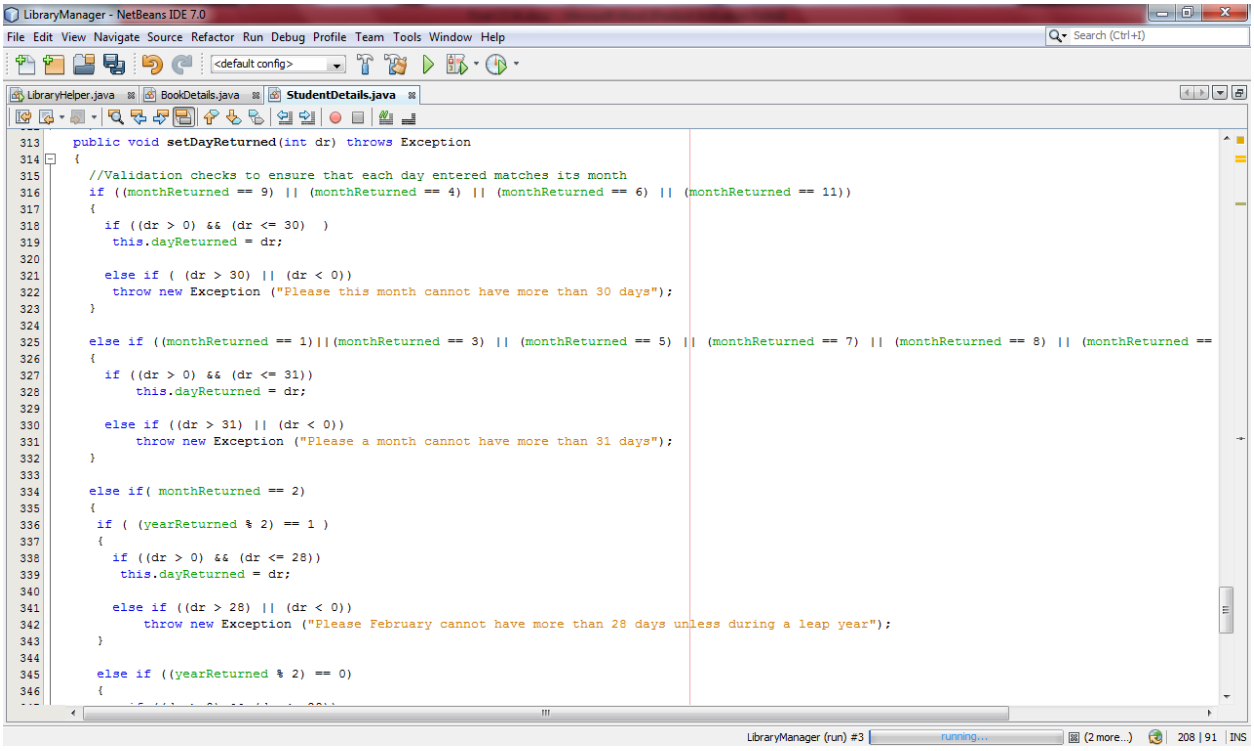
The correct inputs for the year a borrowed book was returned are the same as the correct input for the year a borrowed book was due. Thus, the code for handling error in this field is:

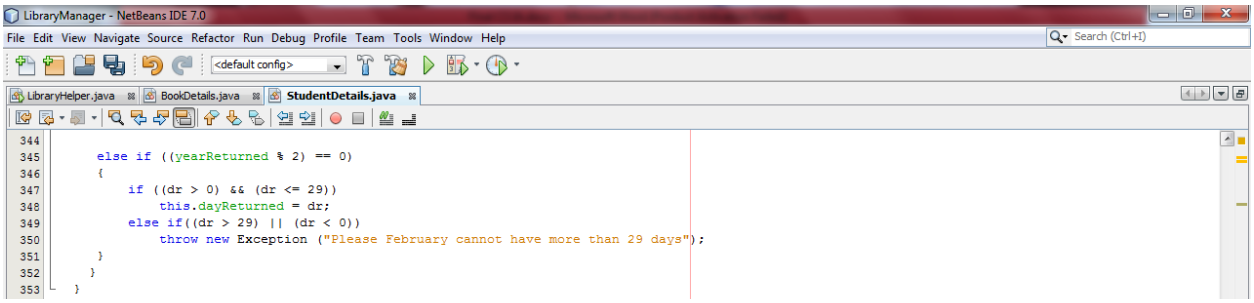


Also the correct inputs for the month a borrowed book was returned are the same as the correct inputs for the month a borrowed book was due. Thus, the code for handling error in this field is:



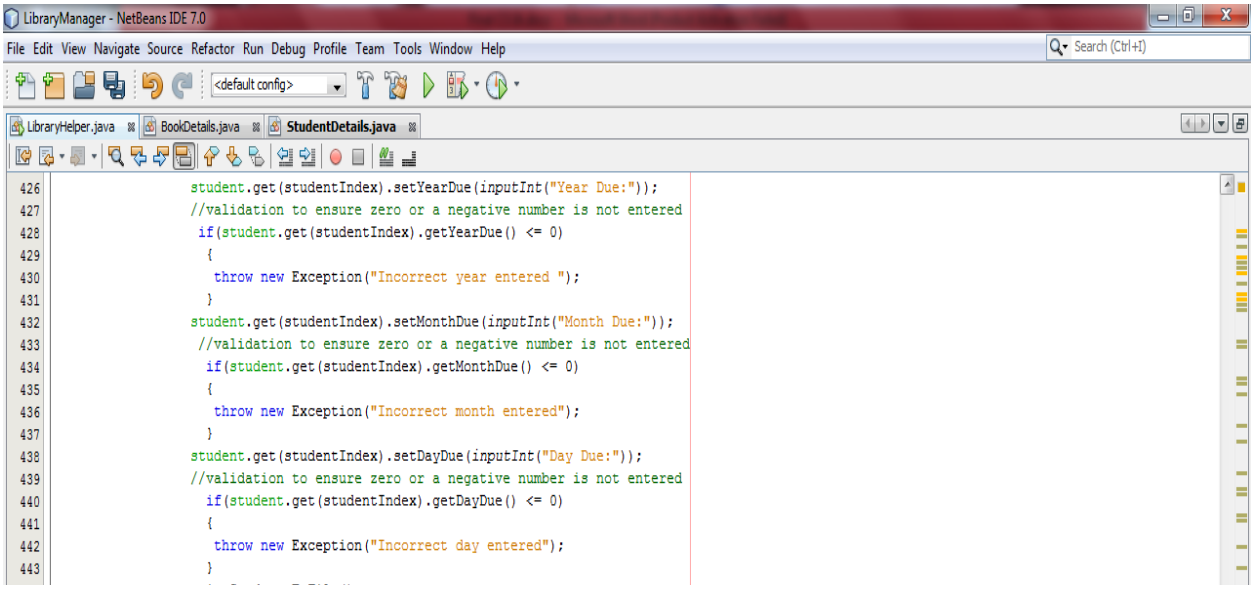
The correct inputs for the day a borrowed book was due is the same as the correct inputs for the day a borrowed book was returned. Thus, the code for handling error in this field is:



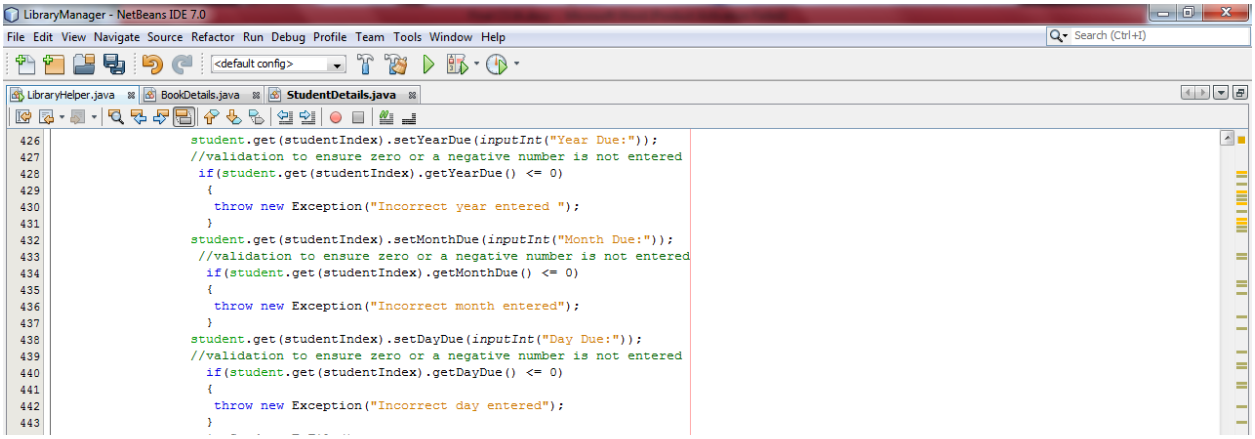


Now, since the yearDue, monthDue, dayDue, yearReturned, monthReturned, and dayReturned fields were made to accept zero in the StudentDetails class, it was important to ensure that the librarian will not zero for these fields when books are being borrowed or returned. The only times these fields will be allowed to be zero will be done by the program itself and not the user. Thus the code for ensuring zero cannot be entered in these fields when books are being returned or borrowed is shown below:

<u>During the borrowing stage</u>



During the returning stage



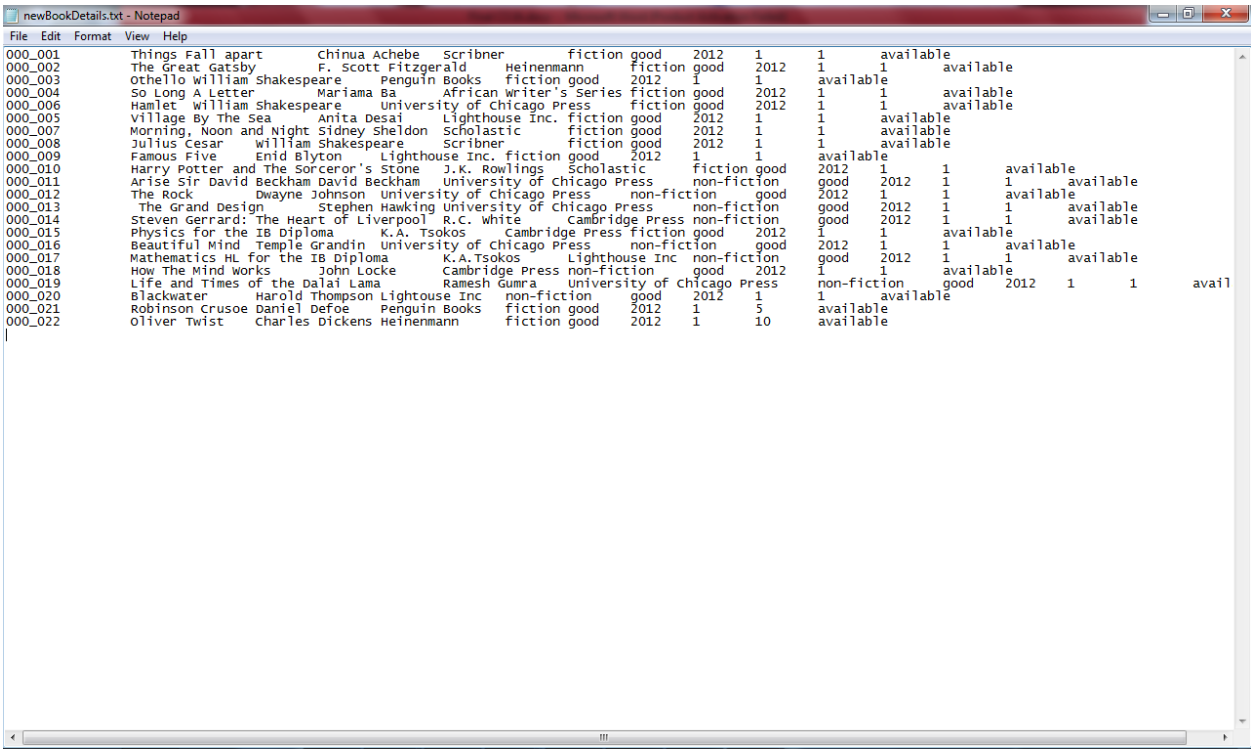
Success of the program

In this section, I will make references to the objectives set in the criteria for success, and display sample outputs of the program that show that the program achieves these objectives.

The program should be able to store data

1. The program should be able to add details of new books to existing database of books during the period the library is being re-stocked.
   Below I will show the program achieves this by adding the details of the book 'The Gods Are Not To Blame', to the existing library database:

The existing library database



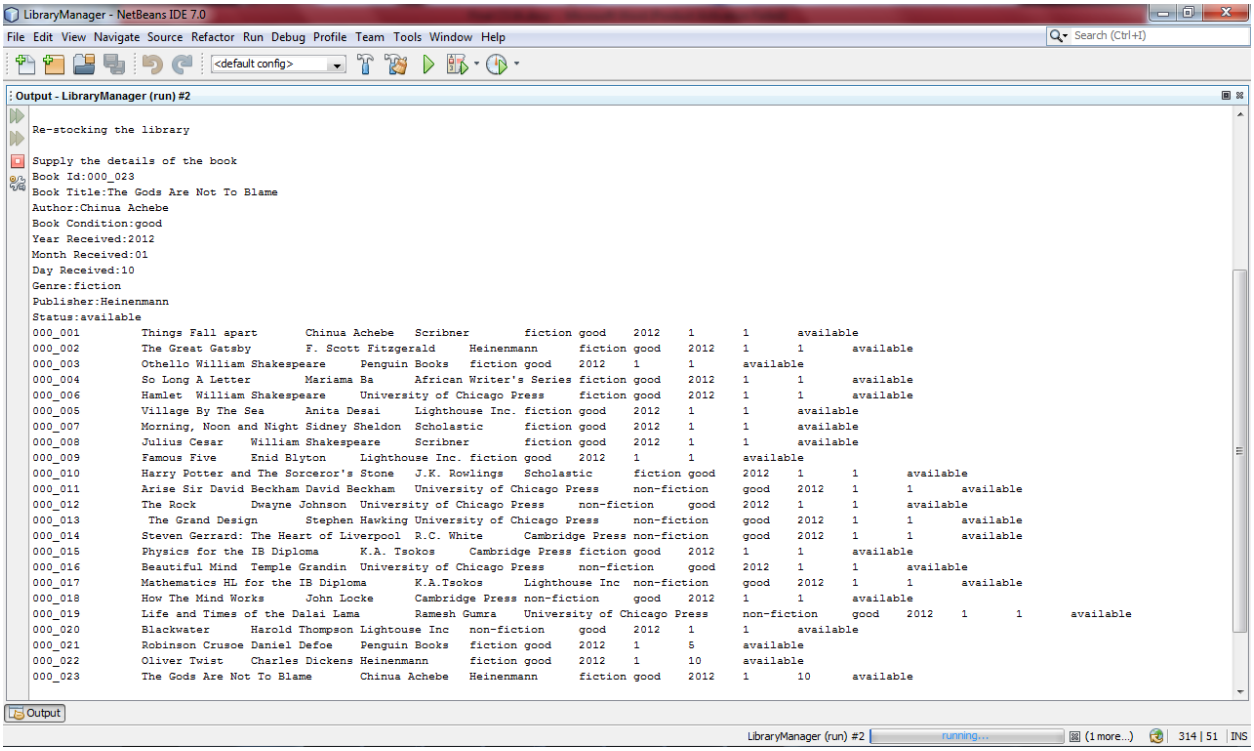
newBookDetails.txt - Notepad

File Edit Format View Help

```
000_001         Things Fall apart       Chinua Achebe   Scribner        fiction good    2012    1       1       available
000_002         The Great Gatsby        F. Scott Fitzgerald     Heinenmann      fiction good    2012    1       1       available
000_003         Othello William Shakespeare     Penguin Books   fiction good    2012    1       1       available
000_004         So Long A Letter        Mariama Ba      African Writer's Series fiction good    2012    1       1       available
000_006         Hamlet  William Shakespeare     University of Chicago Press     fiction good    2012    1       1       available
000_005         Village By The Sea      Anita Desai     Lighthouse Inc. fiction good    2012    1       1       available
000_007         Morning, Noon and Night Sidney Sheldon  Scholastic      fiction good    2012    1       1       available
000_008         Julius Cesar    William Shakespeare     Scribner        fiction good    2012    1       1       available
000_009         Famous Five     Enid Blyton     Lighthouse Inc. fiction good    2012    1       1       available
000_010         Harry Potter and The Sorceror's Stone   J.K. Rowlings   Scholastic      fiction good    2012    1       1       available
000_011         Arise Sir David Beckham David Beckham   University of Chicago Press     non-fiction     good    2012    1       1       available
000_012         The Rock        Dwayne Johnson  University of Chicago Press     non-fiction     good    2012    1       1       available
000_013          The Grand Design       Stephen Hawking University of Chicago Press     non-fiction     good    2012    1       1       available
000_014         Steven Gerrard: The Heart of Liverpool  R.C. White      Cambridge Press non-fiction     good    2012    1       1       available
000_015         Physics for the IB Diploma      K.A. Tsokos     Cambridge Press fiction good    2012    1       1       available
000_016         Beautiful Mind  Temple Grandin  University of Chicago Press     non-fiction     good    2012    1       1       available
000_017         Mathematics HL for the IB Diploma       K.A.Tsokos      Lighthouse Inc  non-fiction     good    2012    1       1       available
000_018         How The Mind Works      John Locke      Cambridge Press non-fiction     good    2012    1       1       available
000_019         Life and Times of the Dalai Lama        Ramesh Gumra    University of Chicago Press     non-fiction     good    2012    1       1       avail
000_020         Blackwater      Harold Thompson Lightouse Inc   non-fiction     good    2012    1       1       available
000_021         Robinson Crusoe Daniel Defoe    Penguin Books   fiction good    2012    1       5       available
000_022         Oliver Twist    Charles Dickens Heinenmann      fiction good    2012    1       10      available
```

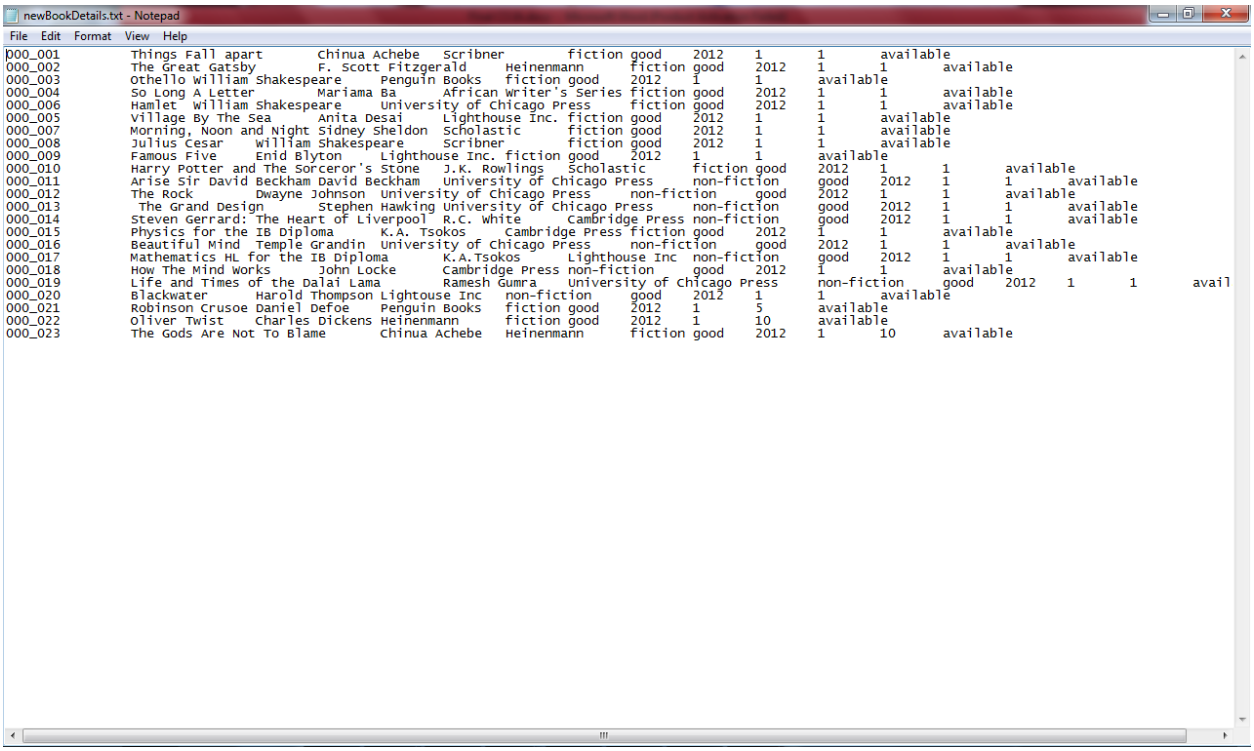
As the diagram shows, there is no book called ‘The Gods Are Not To Blame’ in the newBookDetails file.

Now when I run the program to add the details of these books:



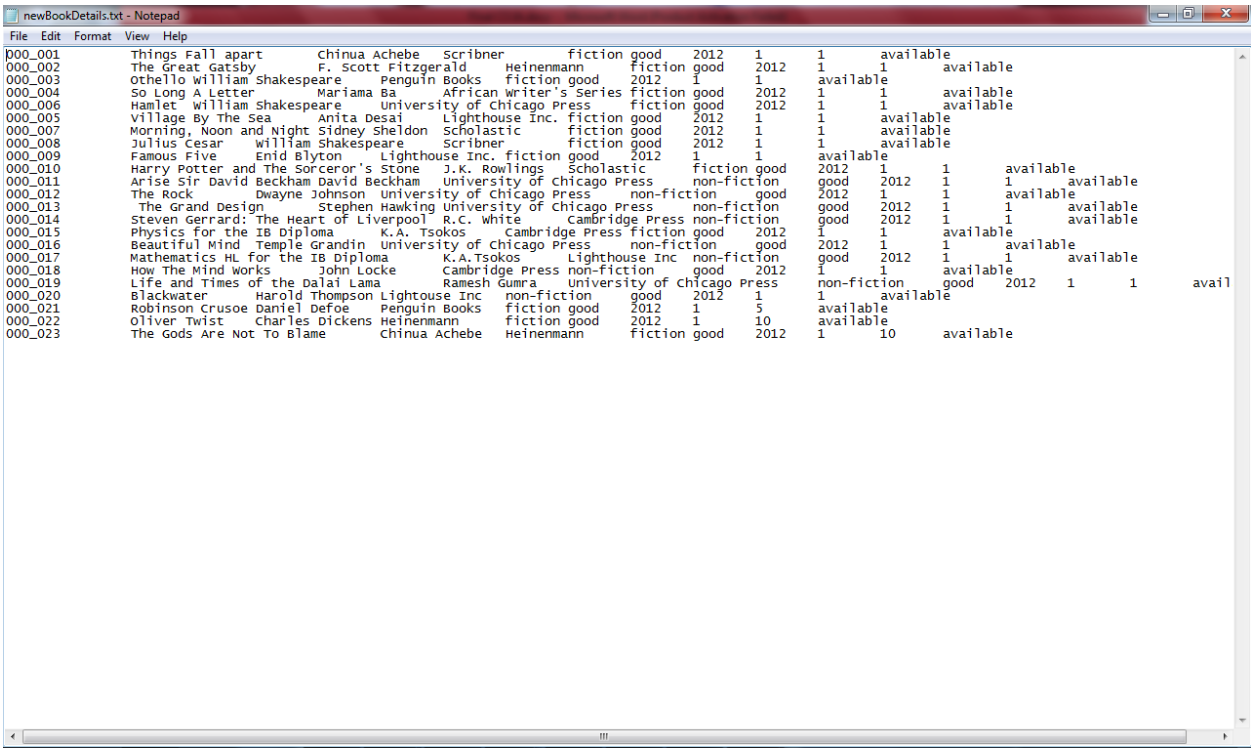
As the diagram above shows, the details of the book ‘The Gods Are Not To Blame’ has been added to the library database, and it is the last book profile in the output of the book profiles currently in the library.

The fact that this book has been added to the library database is also shown in the fact that this books is added to the newBookDetails file. This is shown below:



2. The program should be able to record the following book details: book title, book publisher, book genre, and the date a particular book was received.

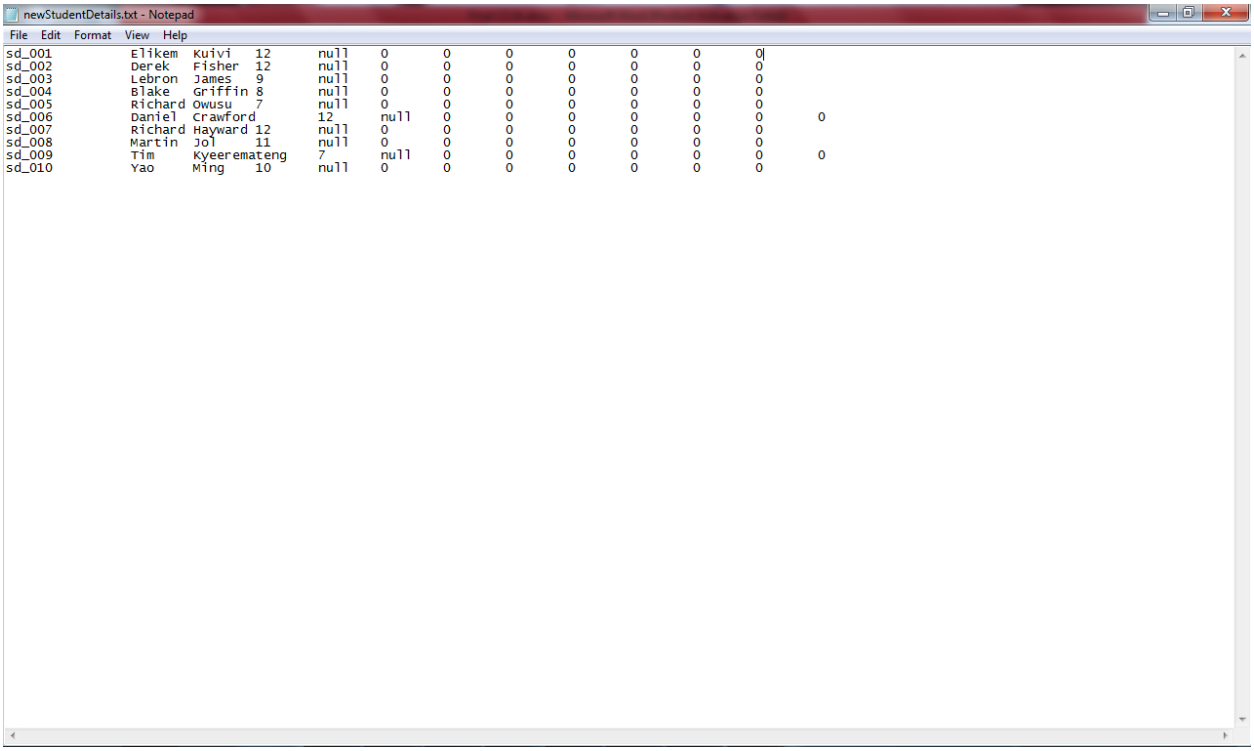
The output below shows this:



As the diagram above shows, the details of all these books a been stored in the newBookDetails file. For example the first profile shows that the title of the book is 'Things Fall Apart', its publisher is 'Scribner', its genre is 'fiction', and that it was received on the 1st of January 2012.

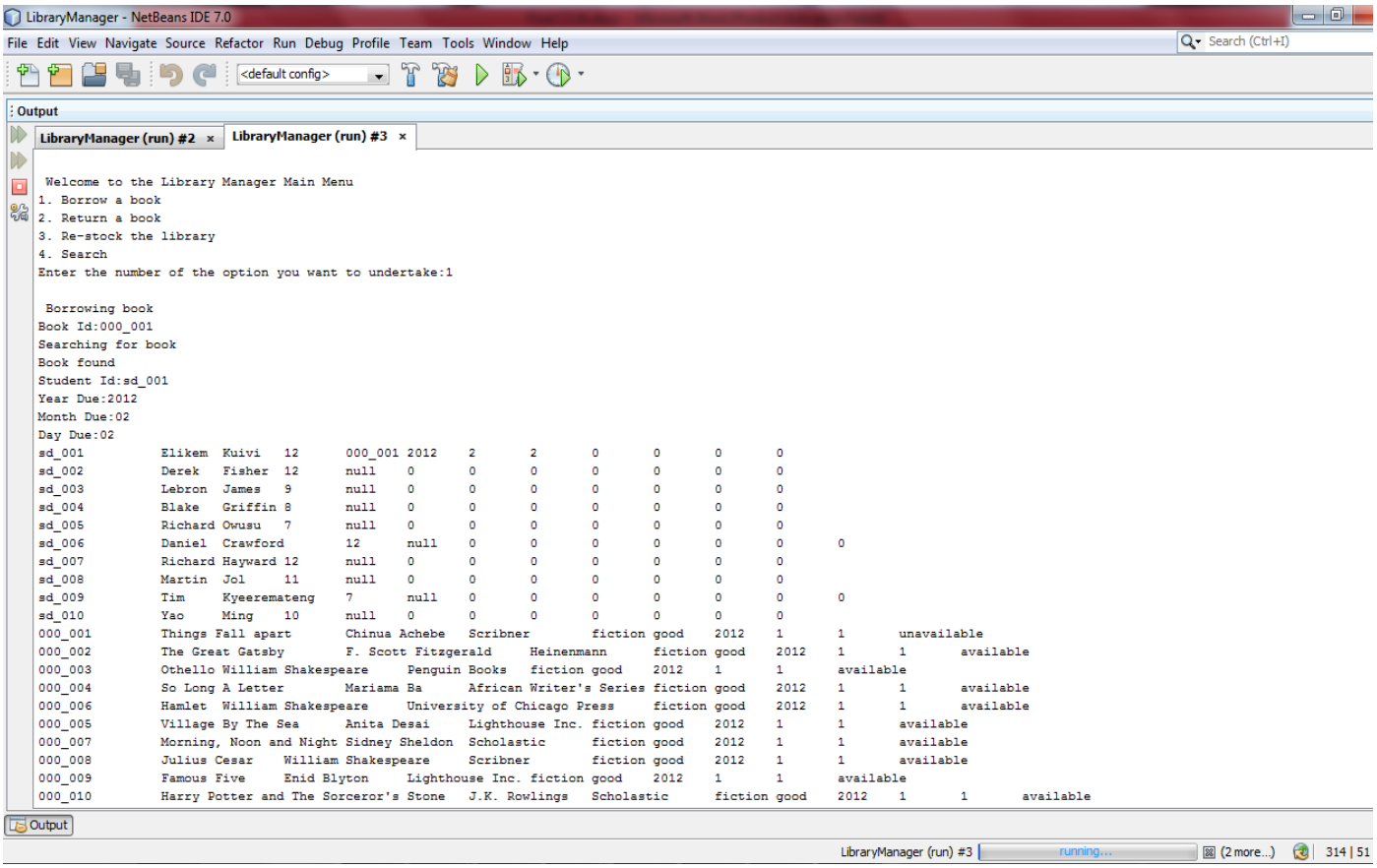
3. The program should be able to store student details in its database

The diagram below shows that the details of students have been stored in the newStudentDetails file:



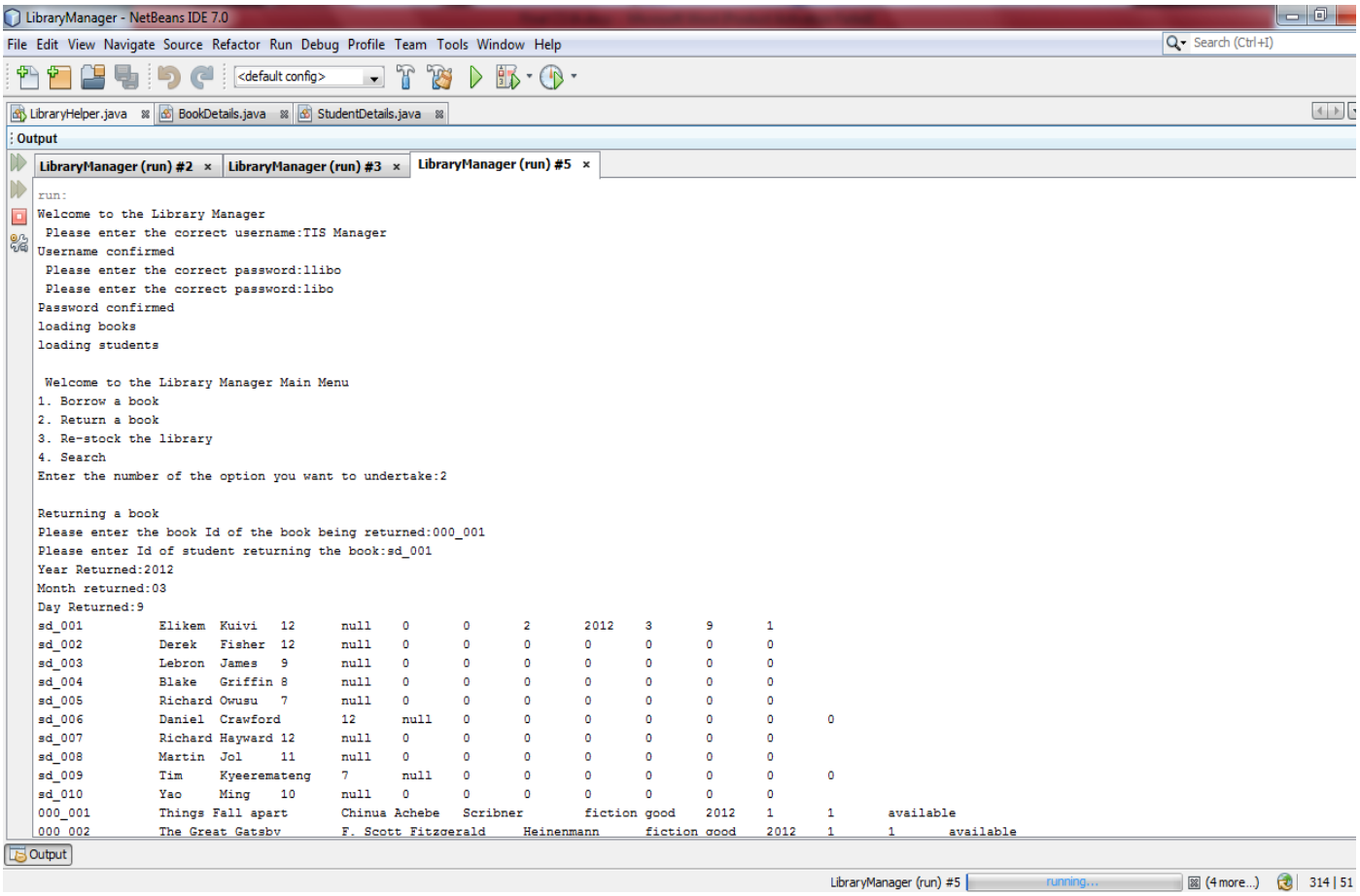
For example, the file shows that the details of a student with Id ‘sd_001’, name ‘Elikem Kuivi’, who is in grade ‘12’, and who has not borrowed any book, has been stored by the program.

4. The program should allow the librarian to input data like, “student Id”, “book Id”, and the date a borrowed book is due, during the stage a book is being borrowed from the library.



As the diagram above shows, the user was able to input that the book with Id ‘000_001’ was being borrowed by the student with Id ‘sd_001’, and that the book is due on the 2nd of February, 2012.

5. The program should allow the librarian to enter the date a book is being returned during the stage a borrowed book is being returned by a student.

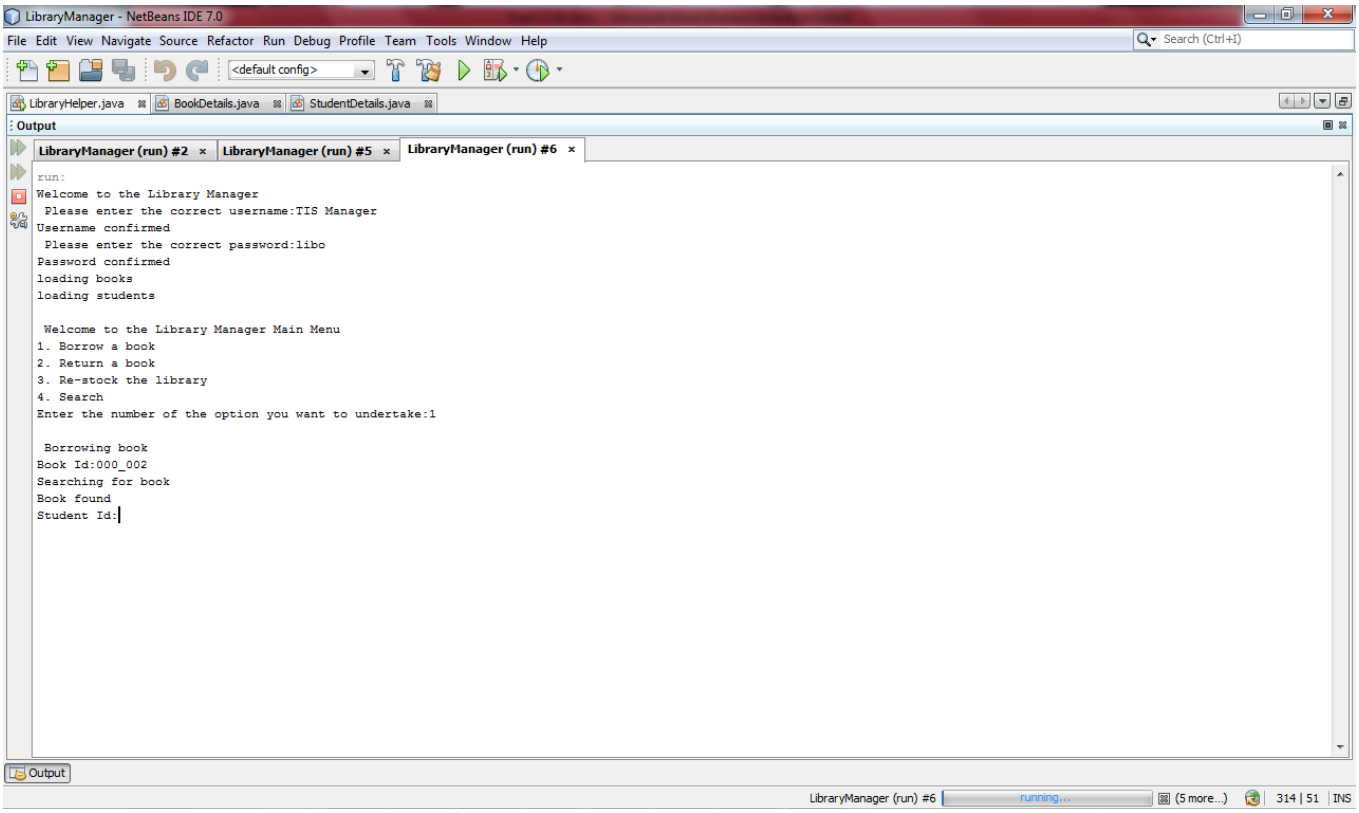


As the diagram above shows, the user was able to input the year, month, and day, the borrowed book was returned. The diagram shows that the user inputted that the user inputted that the student with Id 'sd_001', returned the book with Id '000_001', on the 9th of March, 2012.

The program should be able to allow data to be searched

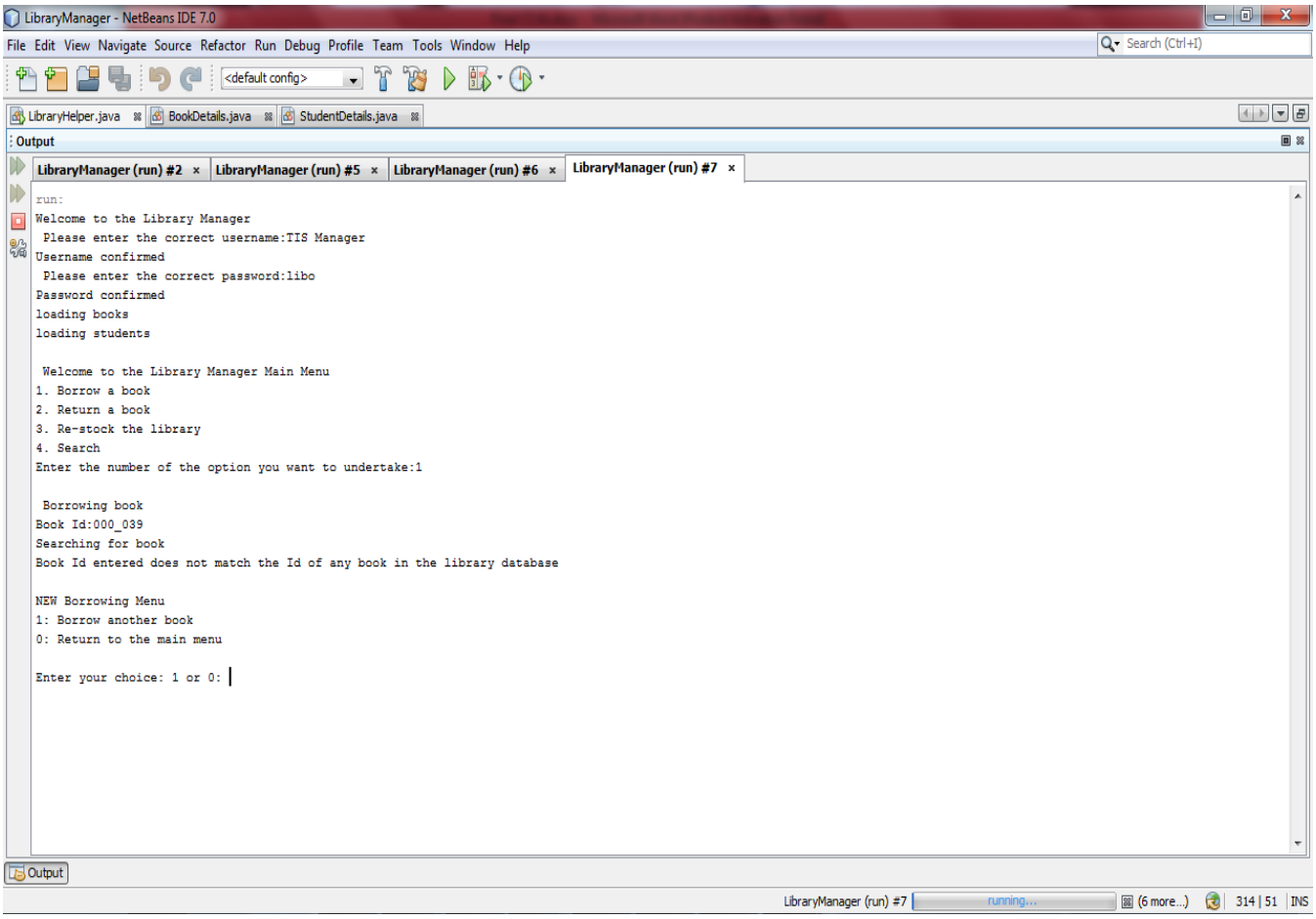
6. The program should allow the librarian to search for specific books in the database during the stage a book is being borrowed

The following will show the scenario when a book to be borrowed has been found to exist in the database, and that the book can be borrowed:



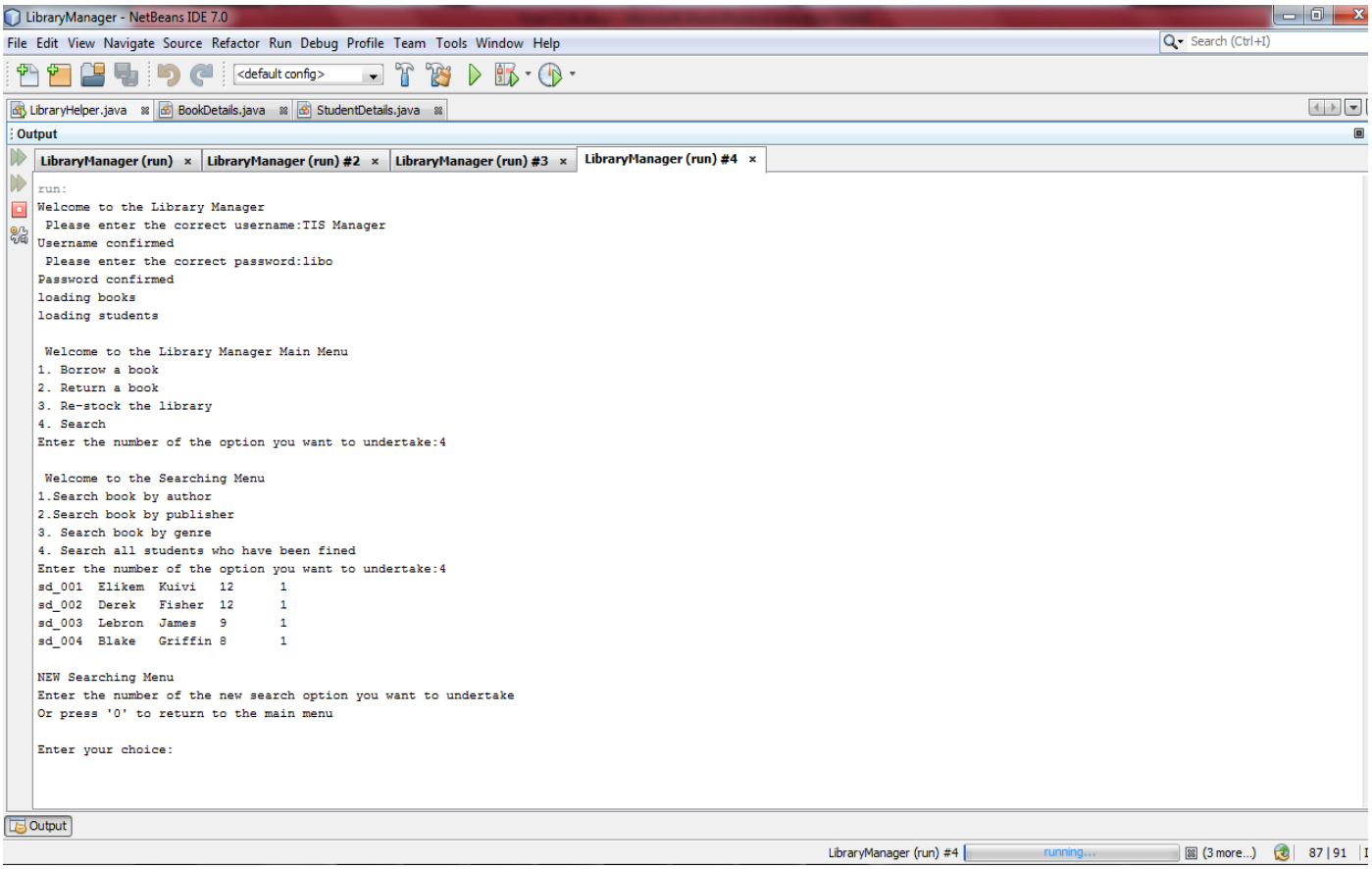
As the diagram above shows, the user inputted the Id of the book '000_002', and the book searched for, and found to exist in the library database. Since the book existed in the database, and was 'available', the user was allowed to input the Id of the student who intended to borrow the book.

However, if the book was searched for, and was not found to exist in the library database, the output would have been:



As the diagram above shows, the program searched for the book Id entered, and did not find it in the database; and, it displayed a message to show this.

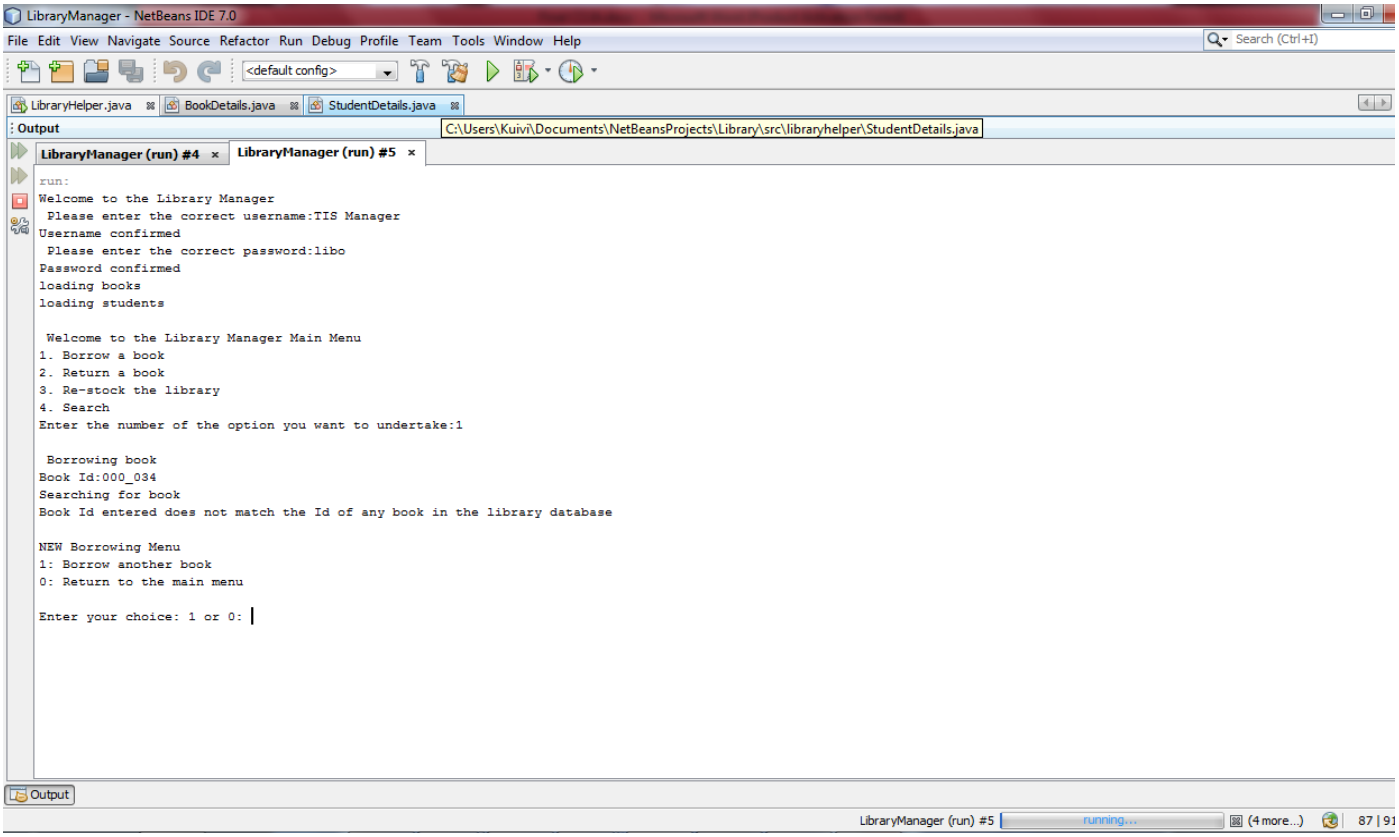
7. The program should allow the librarian to search for names of students who have been fined



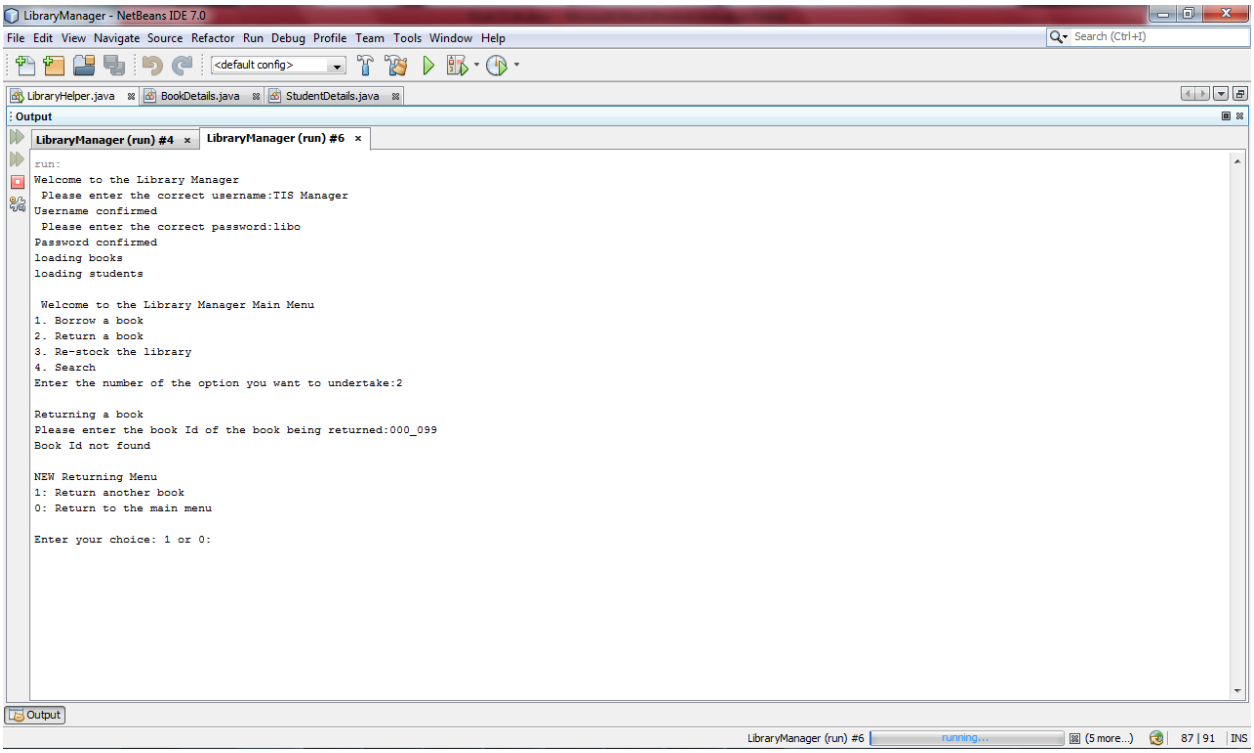
As the diagram above shows, the user chose the option to search for all students who have been fined by the library, and the program search through the studentDetails array list for this, and outputted a list of all the students who were fined by the library. The diagram above shows that 'Elikem Kuivi' (grade 12), 'Derek Fisher' (grade 12), 'Lebron James' (grade 9), and 'Blake Griffin' (grade 8) are all the students who have been fined by the library; and that these students have all accumulated fines of GH₵1.

8. The program should allow the librarian to search through the database to check whether a requested book exists in the library database or not, and should also be able to inform the librarian whether the searched book has been borrowed.

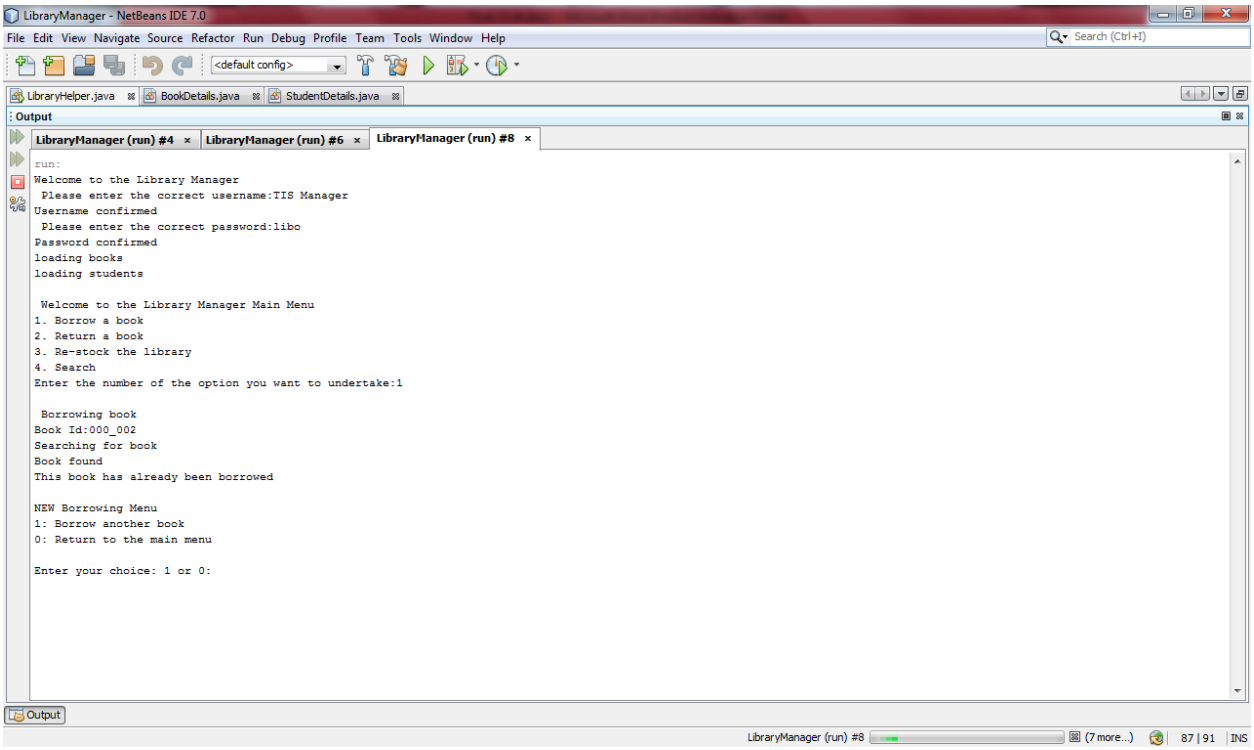
During the borrowing stage, when an inputted book Id does not exist in the library database, the output of the program is:



During the stage a borrowed book is being returned, when an inputted book Id does not exist in the library database, the output of the program is:



In the event, that during the borrowing stage, a requested book has already been borrowed, the output of the program becomes:

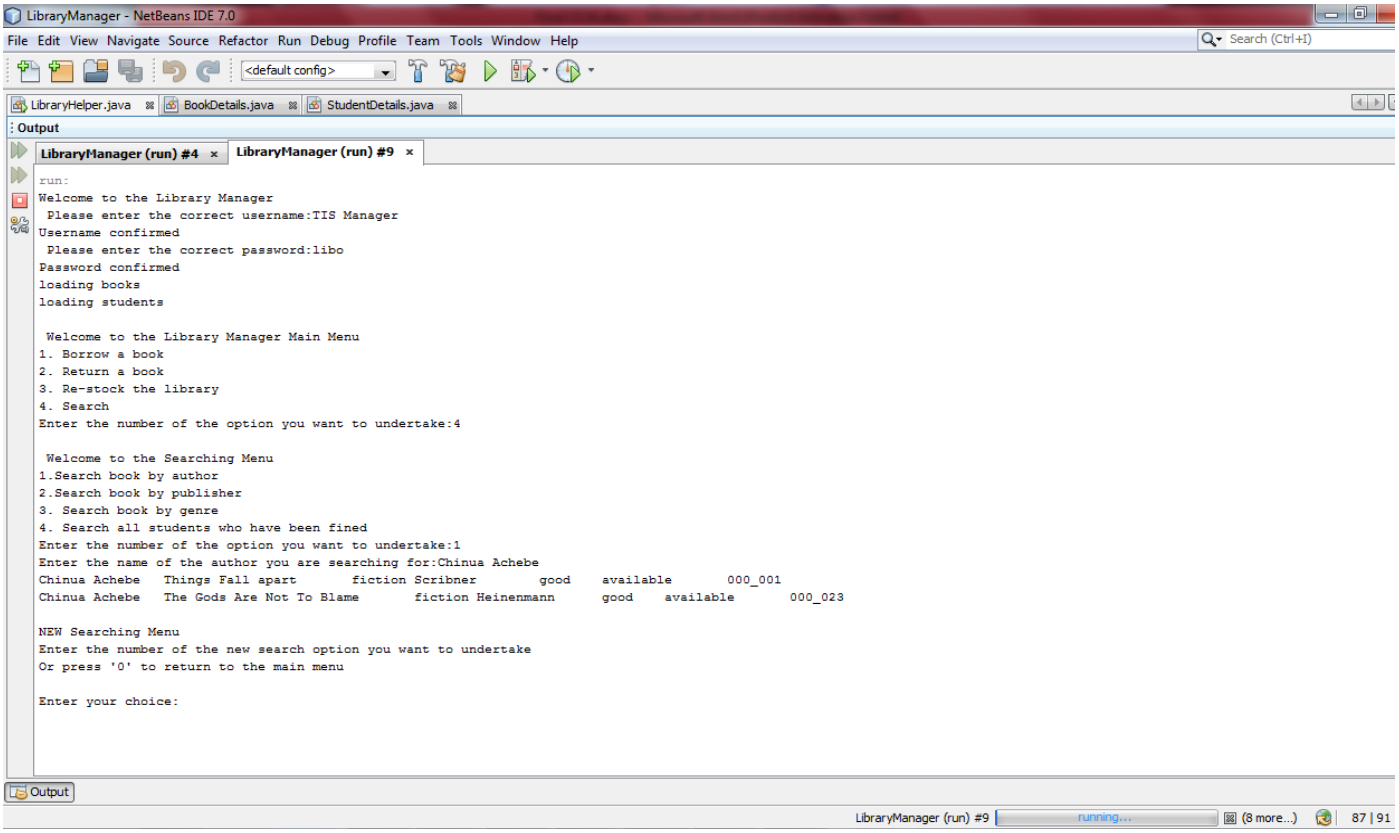


As the diagram above shows, the program has informed the user that the requested book has already been borrowed.

9. The program should allow the librarian to search for books of a particular category (author, publisher, or genre), and return a list of books within that category.

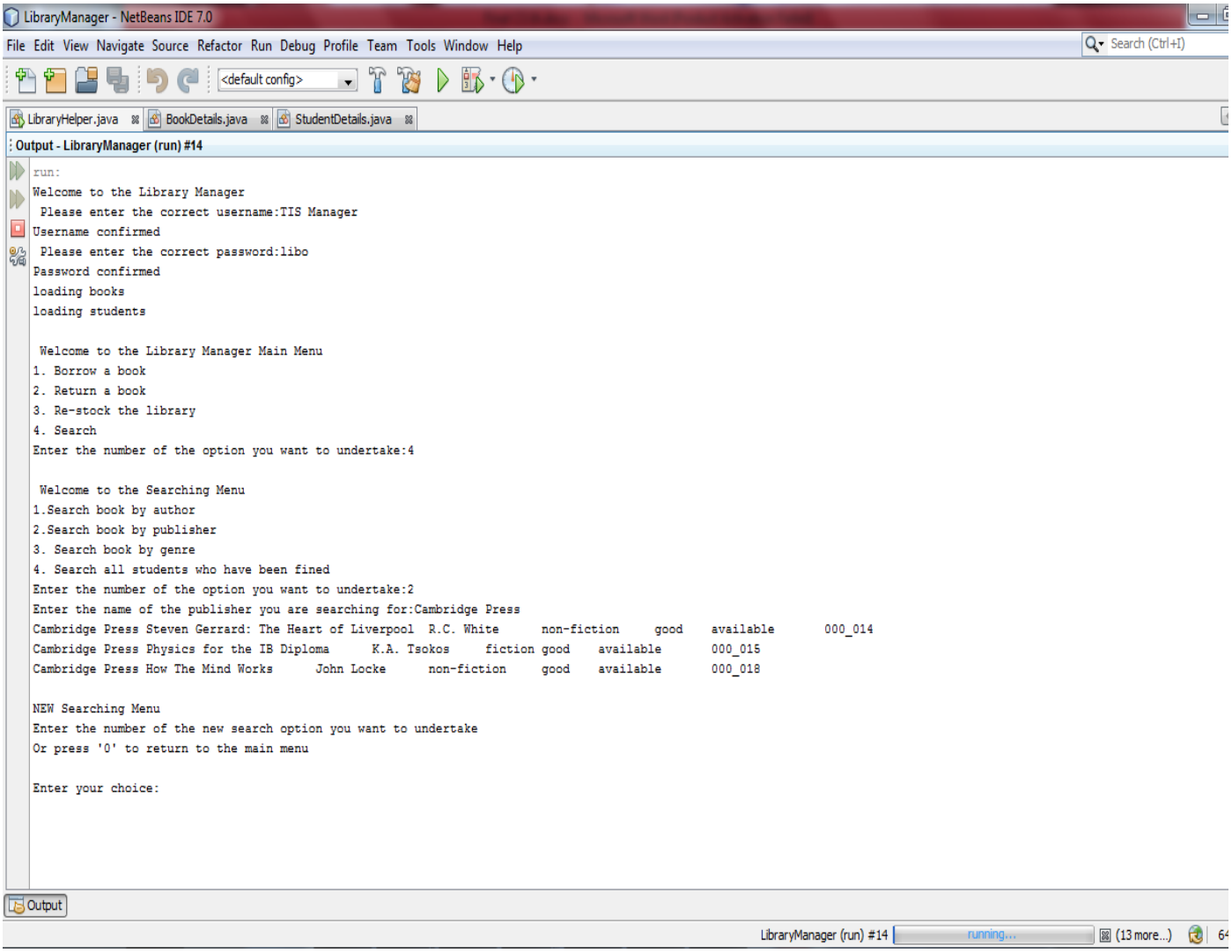
The program gives the user the choice to search for books by their authors, publishers or genres.

A sample output of the user searching for books by their authors is shown below:



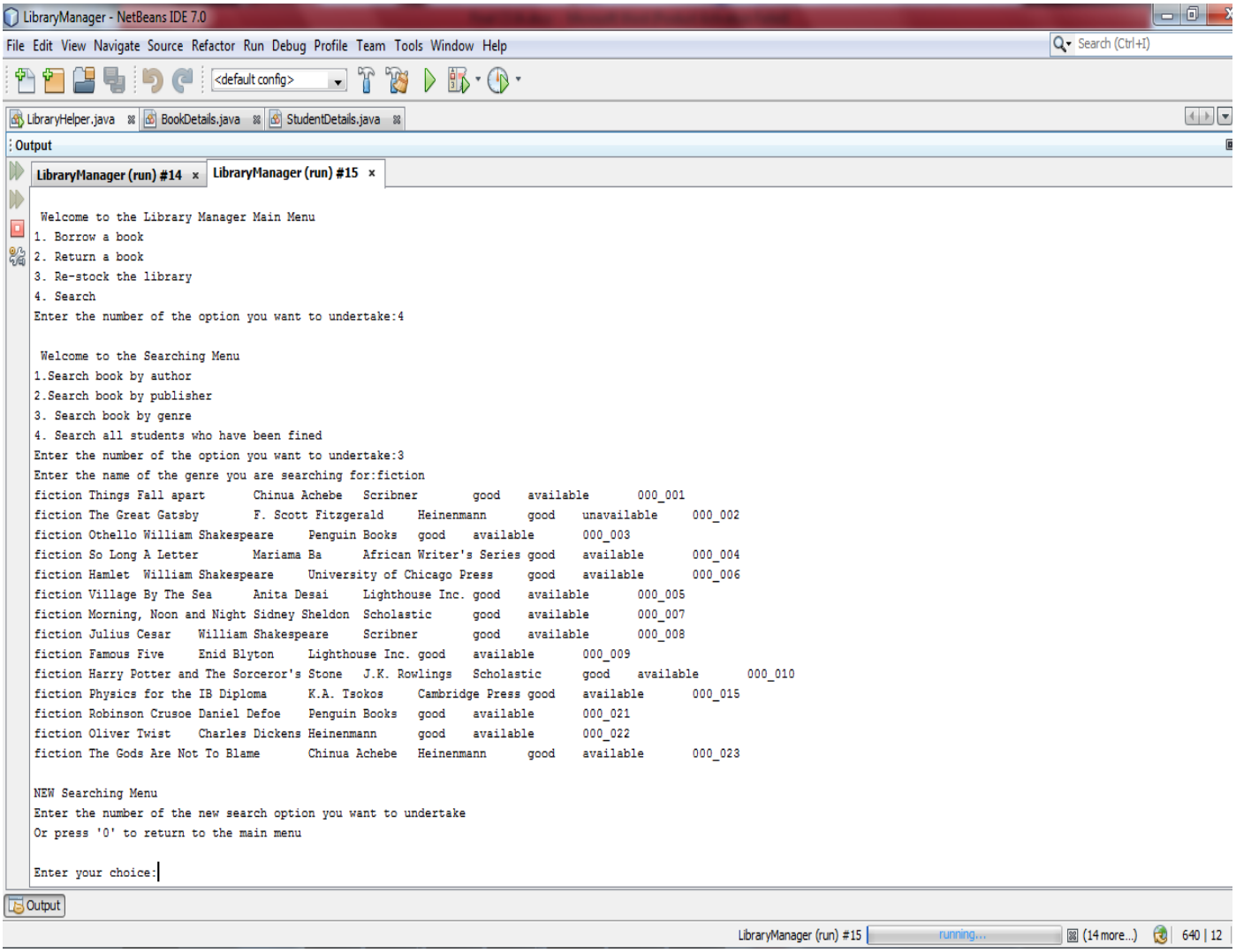
The diagram shows that the user searched for books in the library database that were written by ‘Chinua Achebe’, and the program outputted the list of books in the library database whose authors are Chinua Achebe. The above diagram shows that the books written by ‘Chinua Achebe’ are ‘Things Fall Apart’ and ‘The Gods Are Not To Blame’.

A sample output of the user searching for books by publisher is shown below:



The diagram above shows that the user searched for books published by the ‘Cambridge Prees’, and the program outputted all the books in the library database published by this publisher. The diagram shows that only three books in the library database were published by this publisher; and these books are ‘Steven Gerrard: The Heart of Liverpool’, ‘Physics For The IB Diploma’, and ‘How The Mind Works’.

A sample output of the user searching for books by genre is shown below:



The diagram above shows that the user searched for fictional books in the library database, and that the program outputted a list of all the fictional books in the library database, with their corresponding details. The diagram shows that each line in the list shows the genre, title, author, condition, status, and Id of the book.

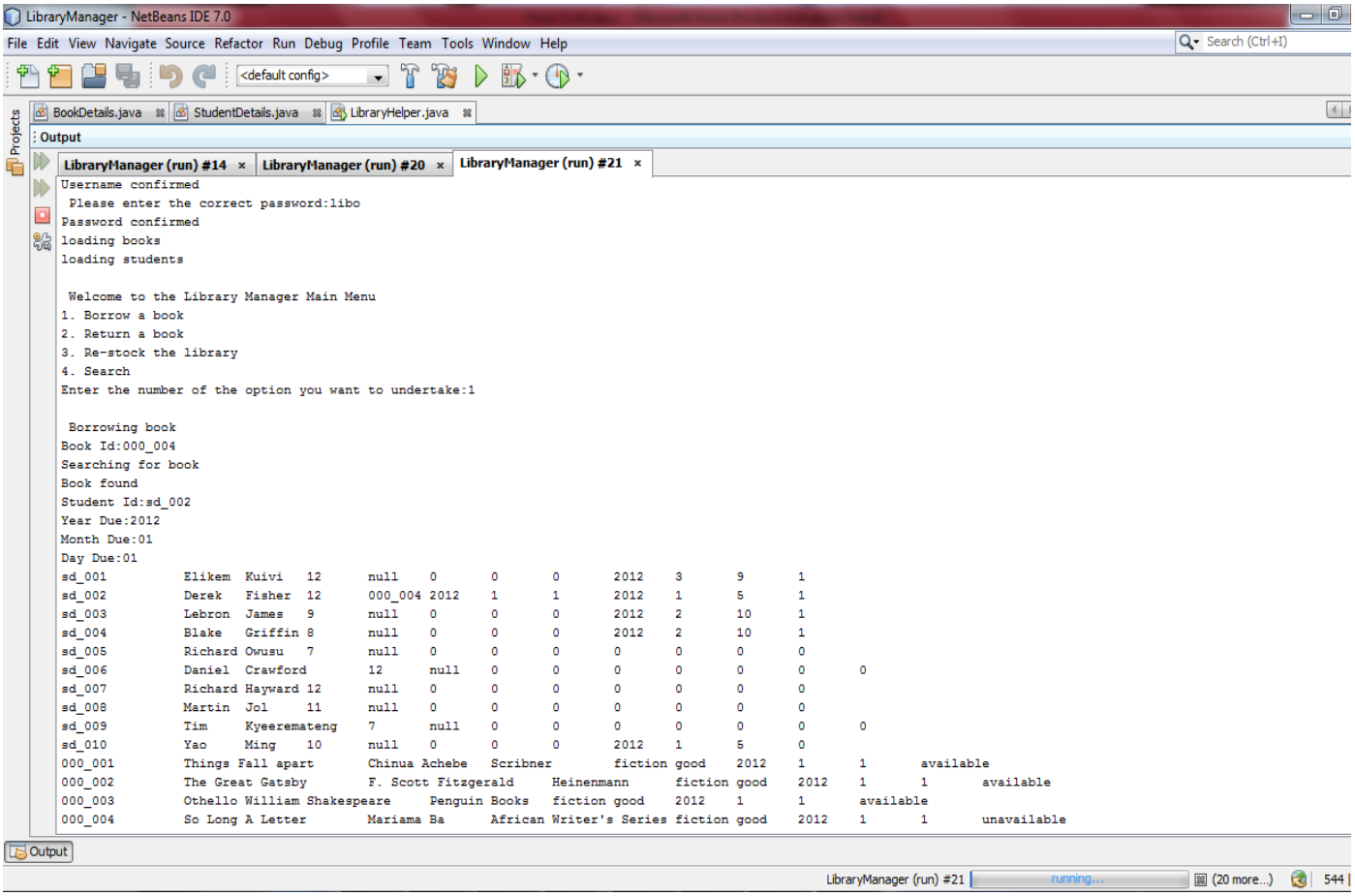
The program should allow data to be edited

10. The program should be able to change the “status” of a book being returned

When a borrowed book is returned, its status should change from ‘unavailable’ to ‘available’. The diagrams below show this:

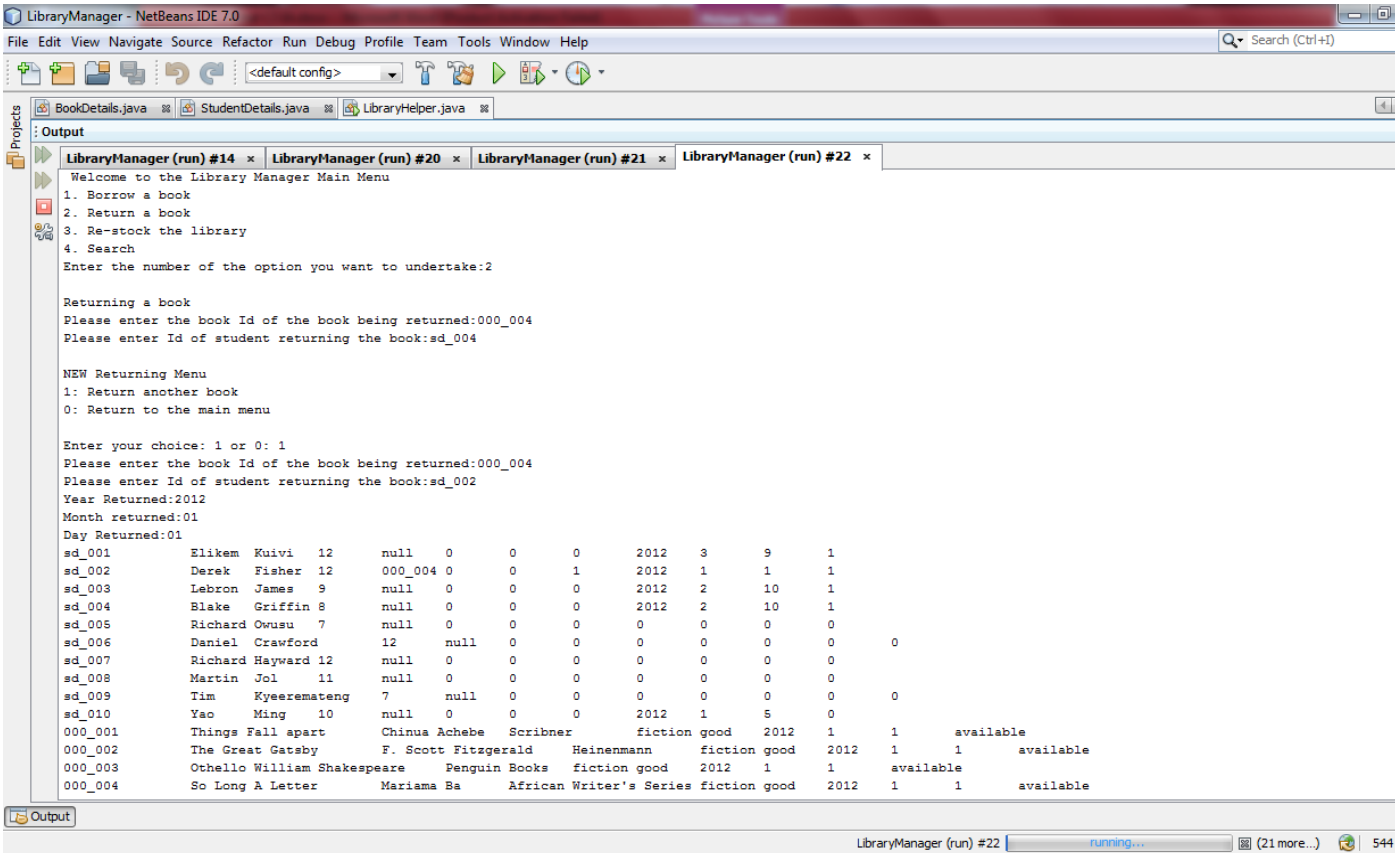
For example, before the book with Id ‘000_004’ is borrowed, its status should be available. This is shown in the newBookDetails file:

Now if this book is borrowed, its status changes to ‘unavailable’. The diagram below shows this:



As the diagram above show, the status of the book ‘000_004’ (the last data record in the screenshot above), has changed to ‘unavailable’.

However, when this book is returned, its status changes to ‘available’:

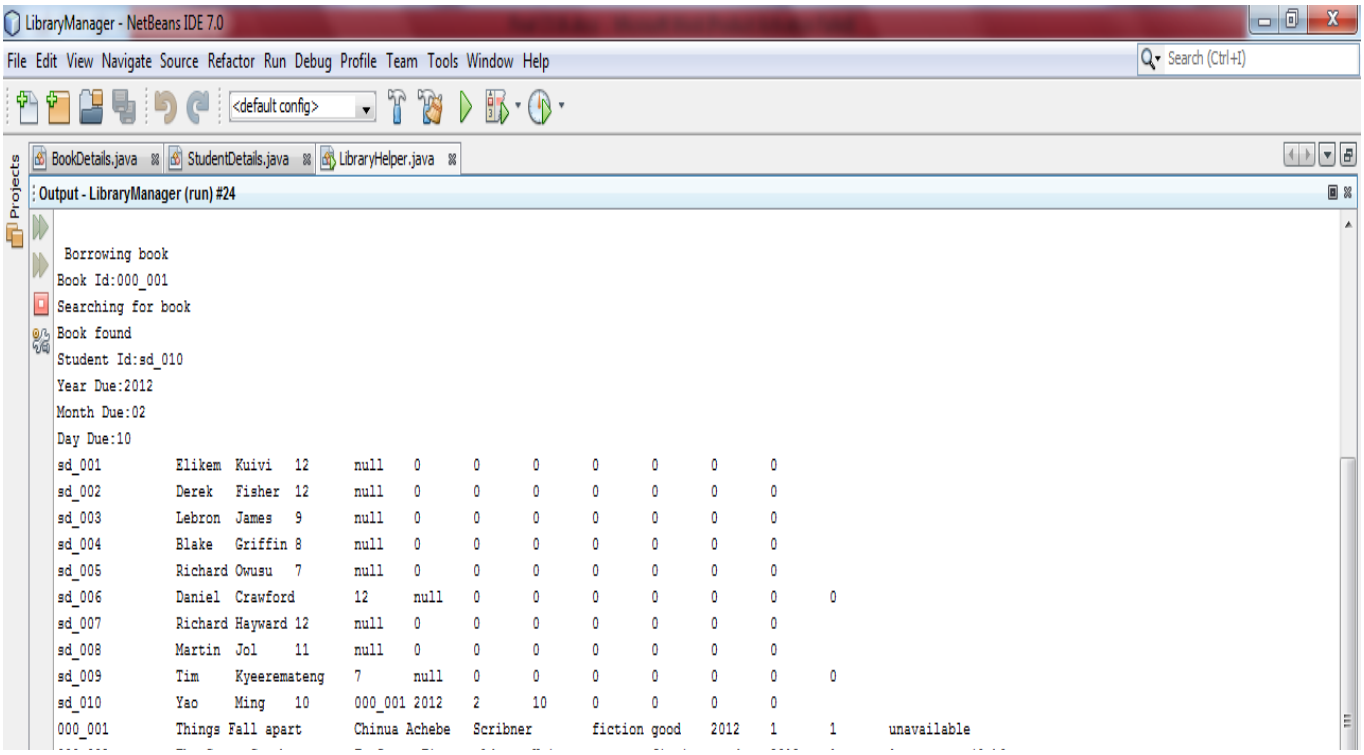


This change in the book’s status is shown in the newBookDetails file:

The program should be able to perform calculations

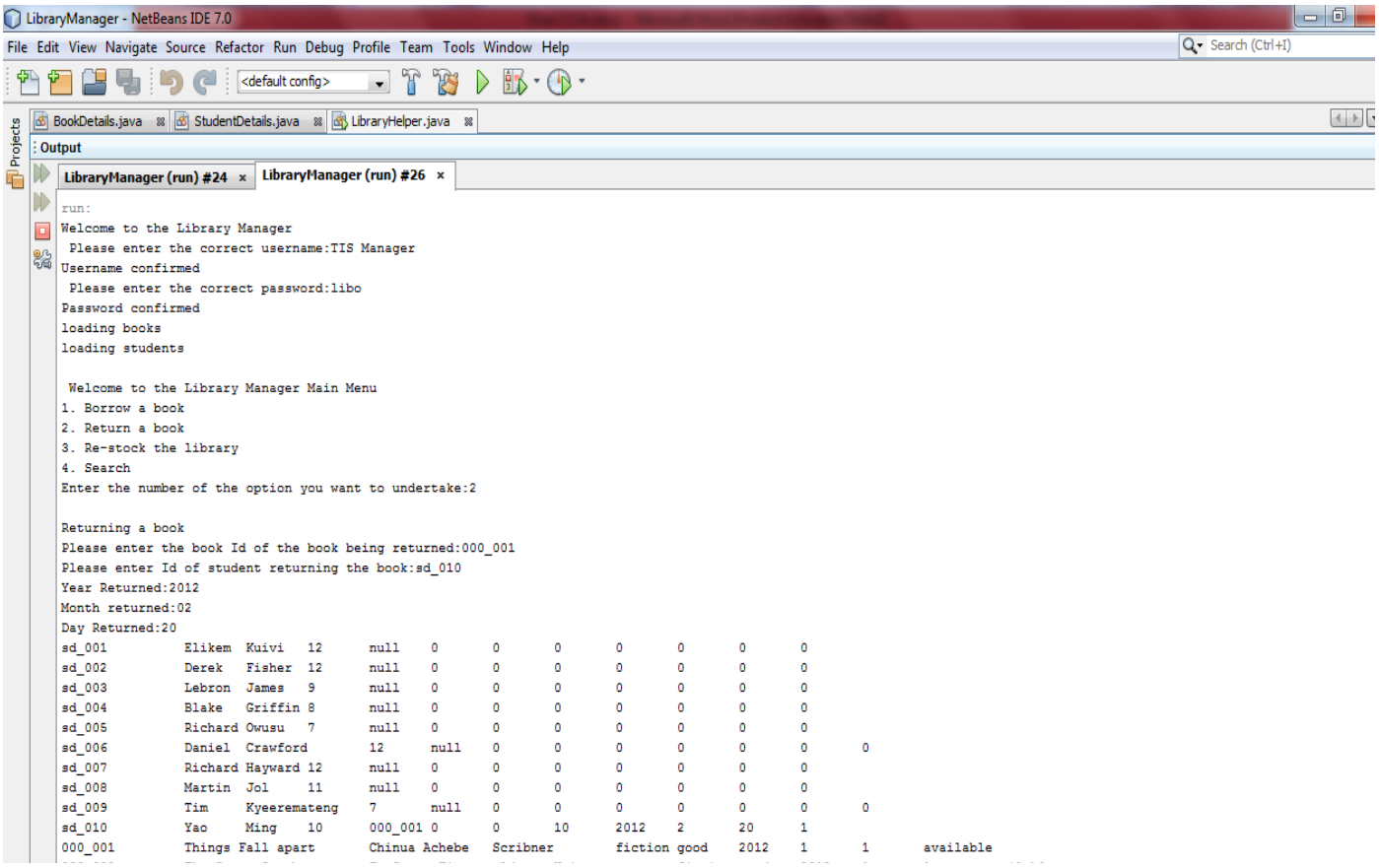
11.The program should be able to check if the book was returned after its due date; and if the book was returned when it was overdue, the program should be able to calculate the total fine the student has accumulated.

Thus in the event that a student borrowed book:



The diagram above shows that the student 'Yao Ming' (sd_010) has borrowed the book 'Things Fall Apart' (000_001), and that the book is due on the 10th of February 2012.

Thus when this student returns the book on the 20th of February 2012, the program adds 1 to this student's tab. Therefore, since this student's accumulated fined is zero, the students accumulated fine will become 1 when the fine is added to this student's fine tab. The ability of the program to perform these calculations is shown below:



As the diagram above shows, the program was able to calculate that the book was returned when it was overdue, and therefore adds 1 to the student's total fine section. Thus, as the diagram above shows, the student "Yao Ming's" total fine now becomes 1, as expected.

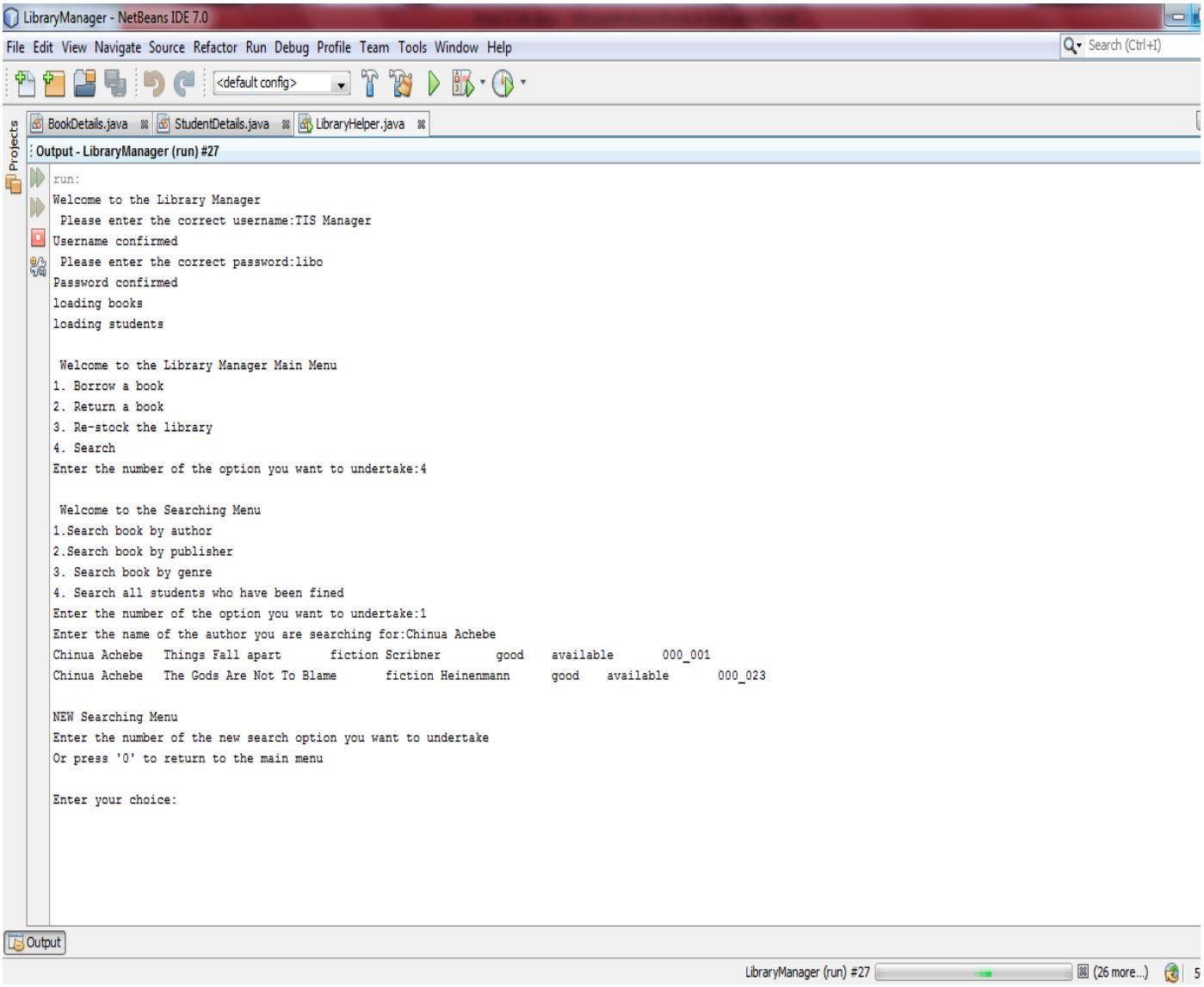
Reliability objectives

12. The interfaces of the new system should be user-friendly so that the librarian can easily use the program and find it more comfortable to use than the old system

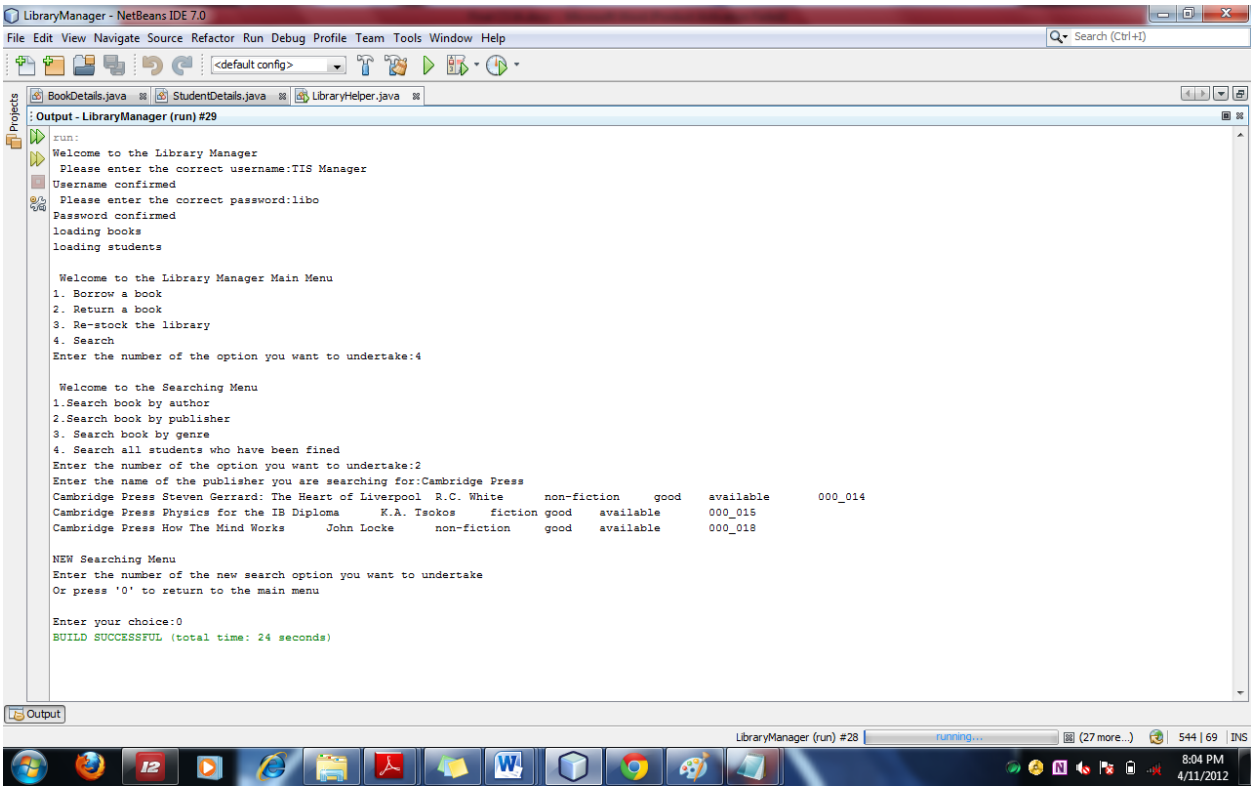
The evidence for this objective was already discussed in the section that talked about user usability.

13. Output should also be easy to read since this will also make the librarian find the program easier to use than the old system

Evidence that this objective was achieved is the fact that users can see the instructions of the program at each stage, and act accordingly. A sample session of the program being used by a user is shown below as evidence that this objective was achieved:



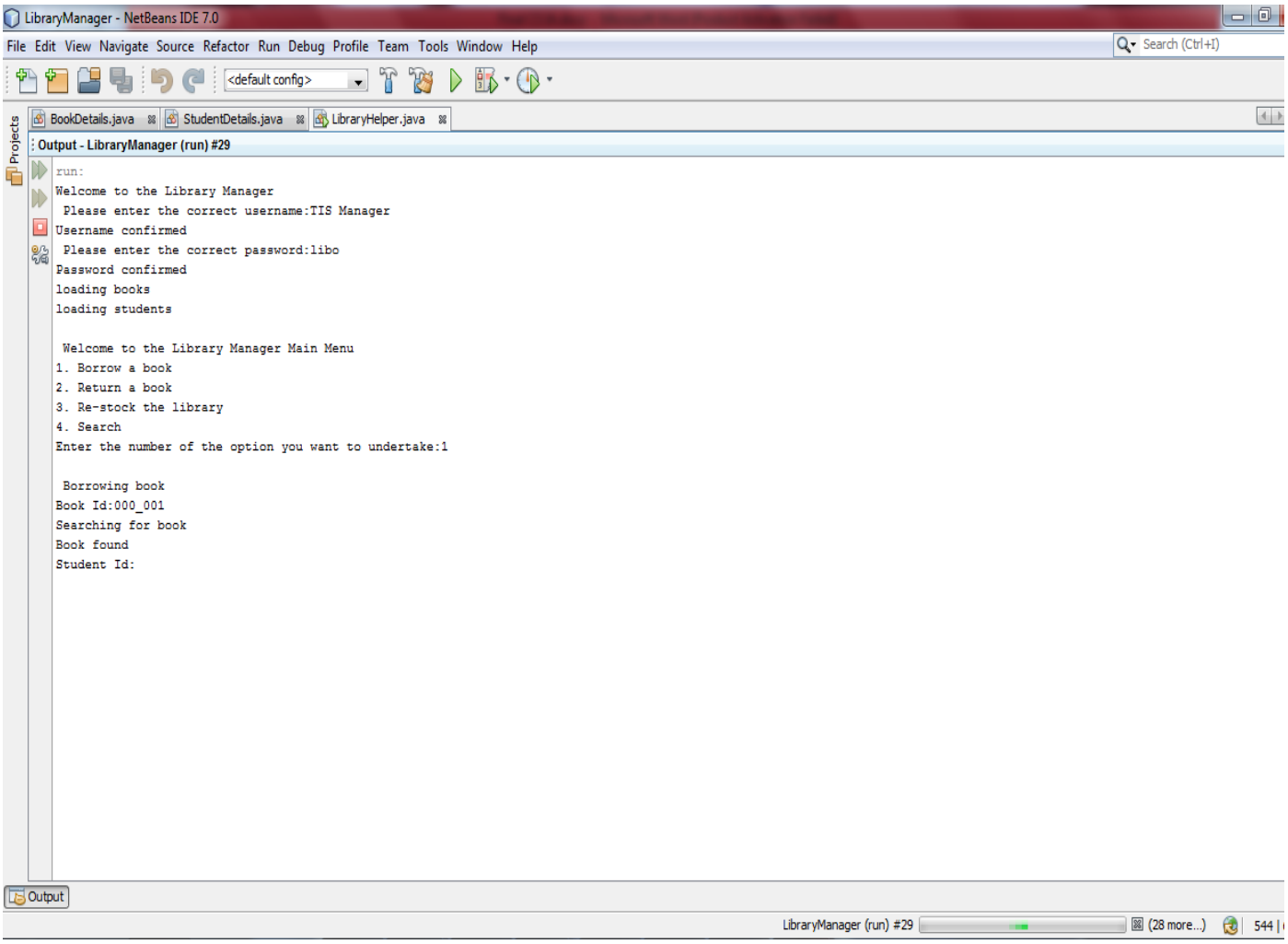
14. Searches should be fast so that the librarian and students alike will not find this program as time-consuming as the old system



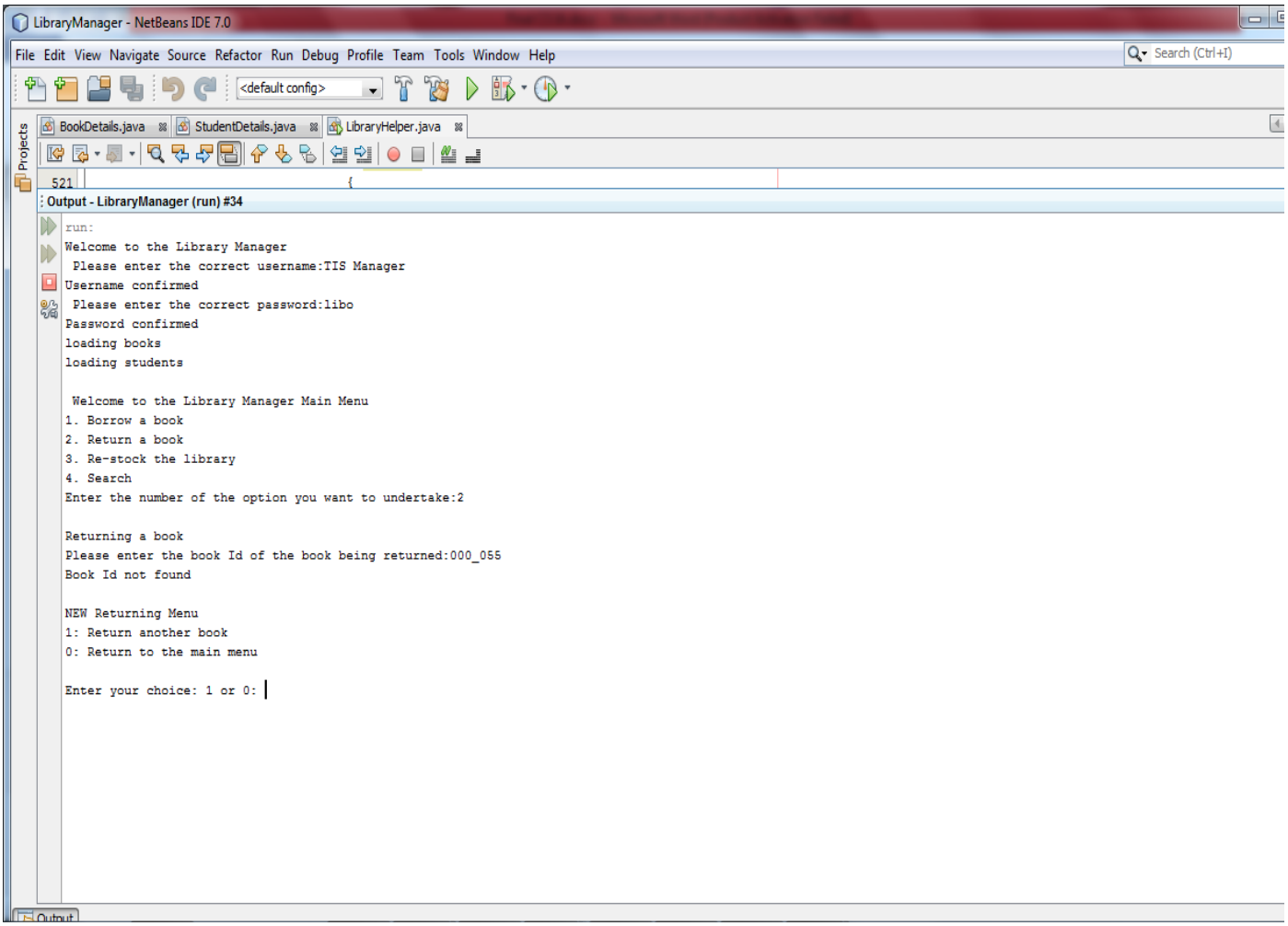
The diagram above shows the instance the user searched for all the books in the library database that were published by 'Cambridge Press'; and, as the above diagram shows, this activity took only 24 seconds, thus it can be said that the program has achieved the above objective since 24 seconds is faster than it would have taken for the librarian to make this search.

15. During the borrowing and returning stages, the program should allow the librarian to search for books in the library database using book Id's instead of book names

Below is a sample output of the program searching for books using book Id's during the borrowing stage:



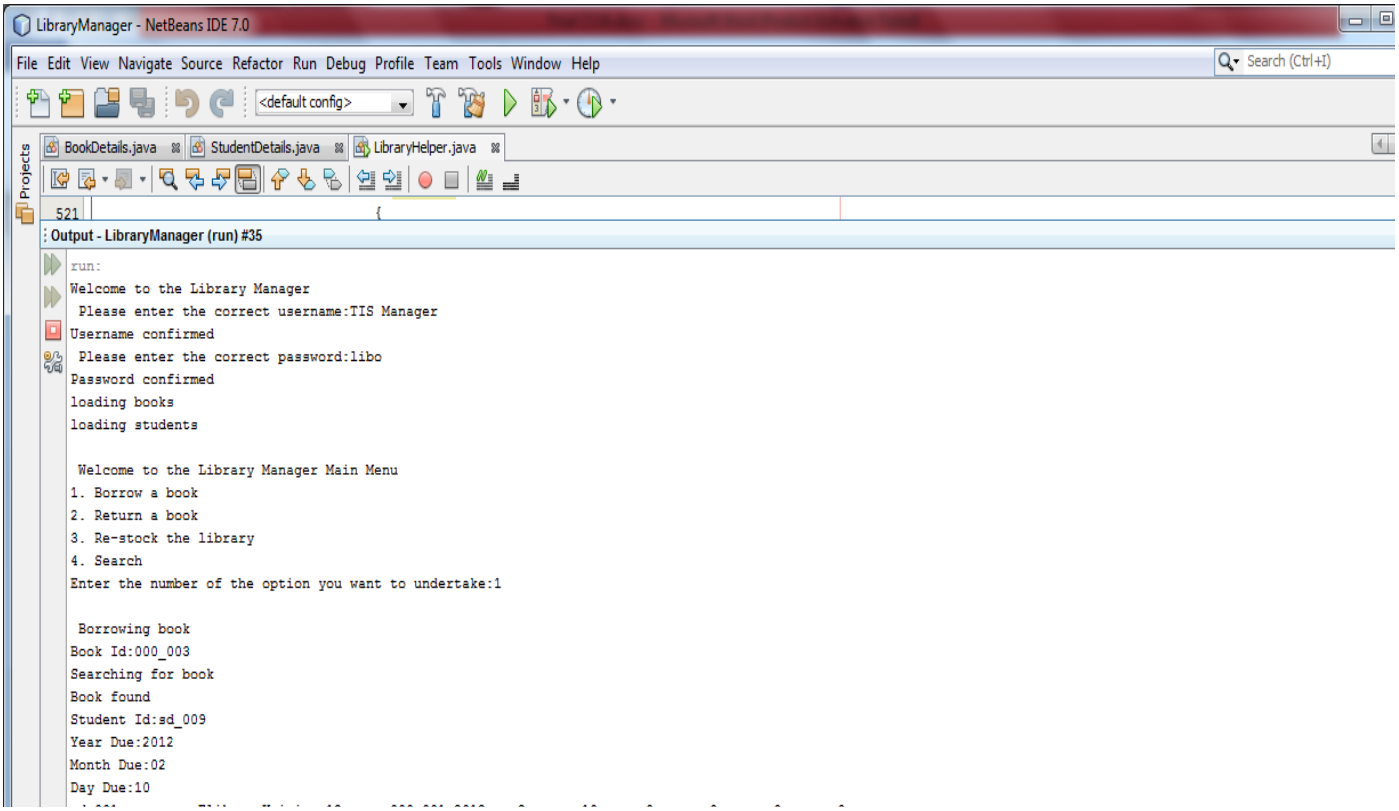
Now, here is a sample output of the program searching for books using book Id's during the returning stage:



The diagram above shows that the inputted book Id was searched for, and was not found.

16. During the borrowing stage, the program should allow the librarian to enter the due dates of books to be borrowed.

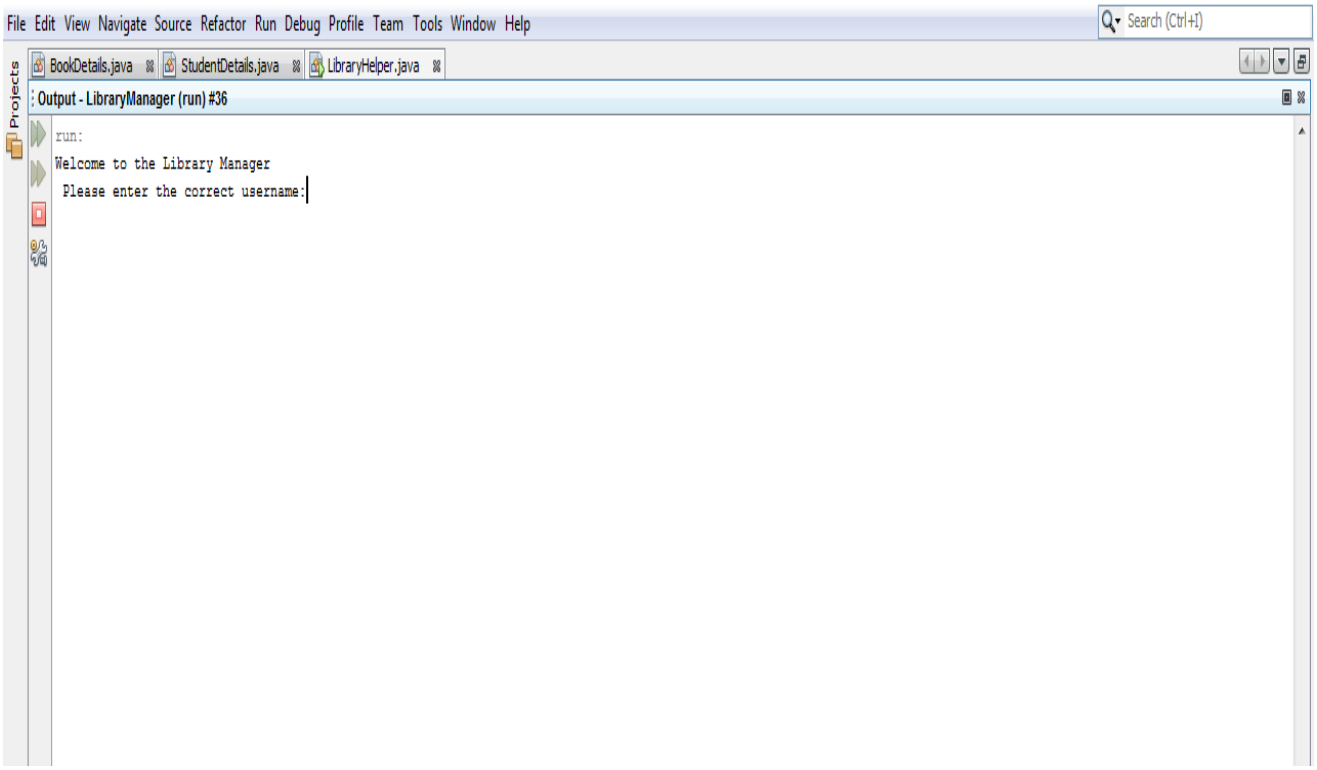
Evidence the program allows this is shown below:



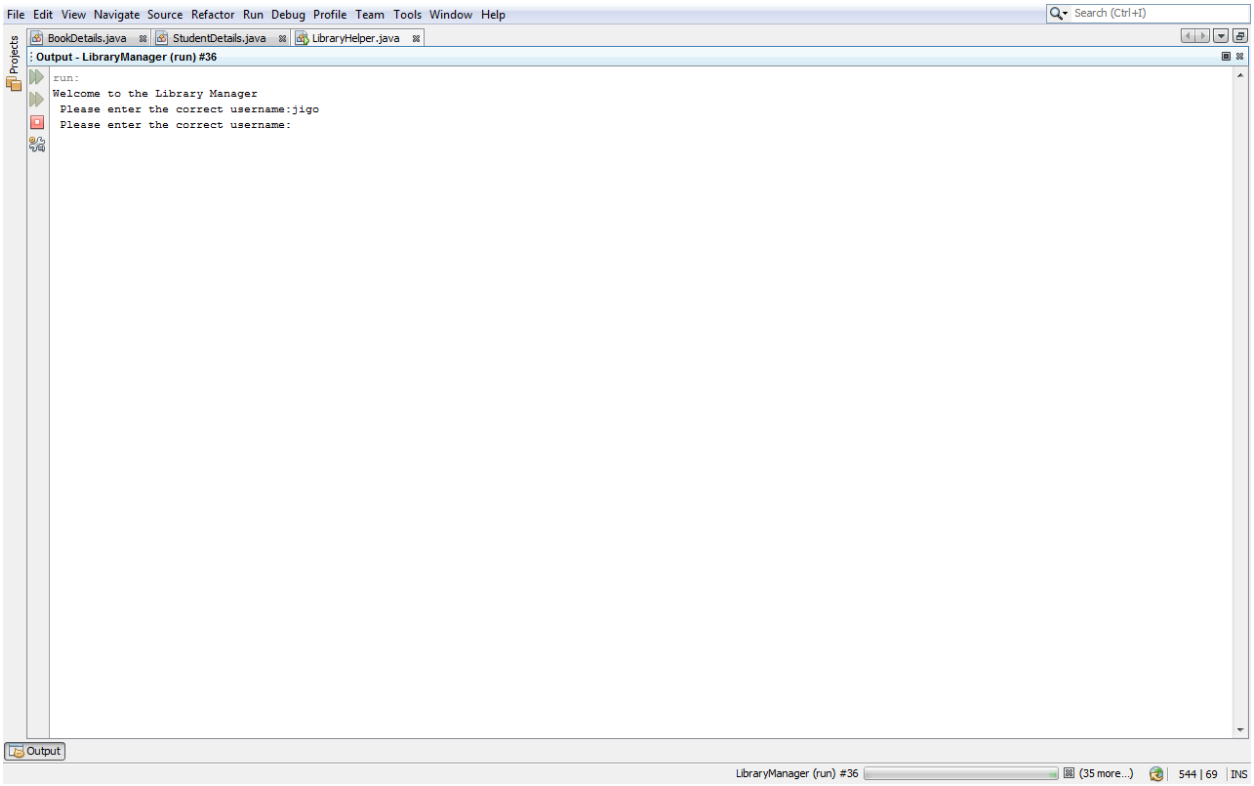
Part D- Test Output

The following section will show a sample run of the program at different stages, and this will include input and output at the different stages of the program.

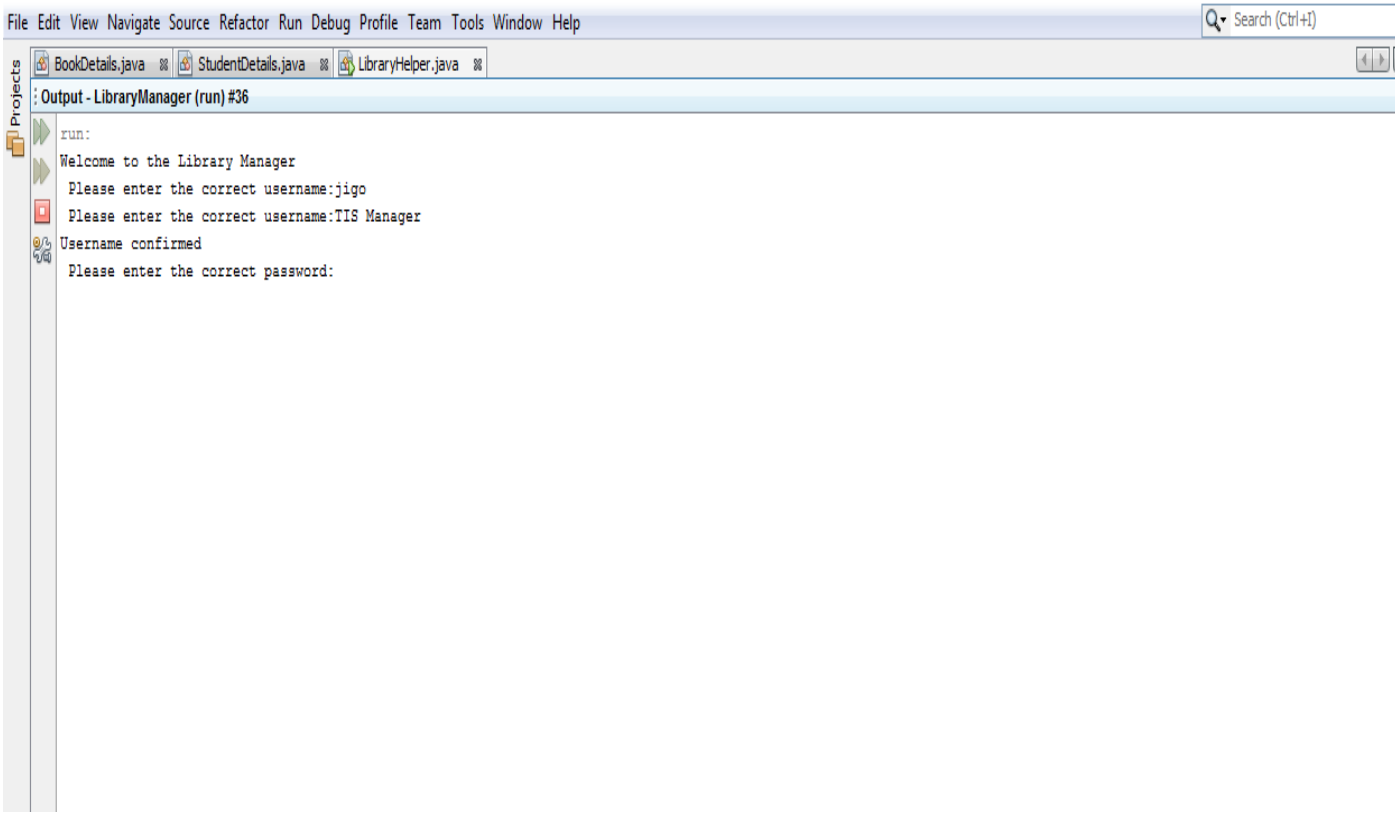
The program begins by asking the user to input the correct username:



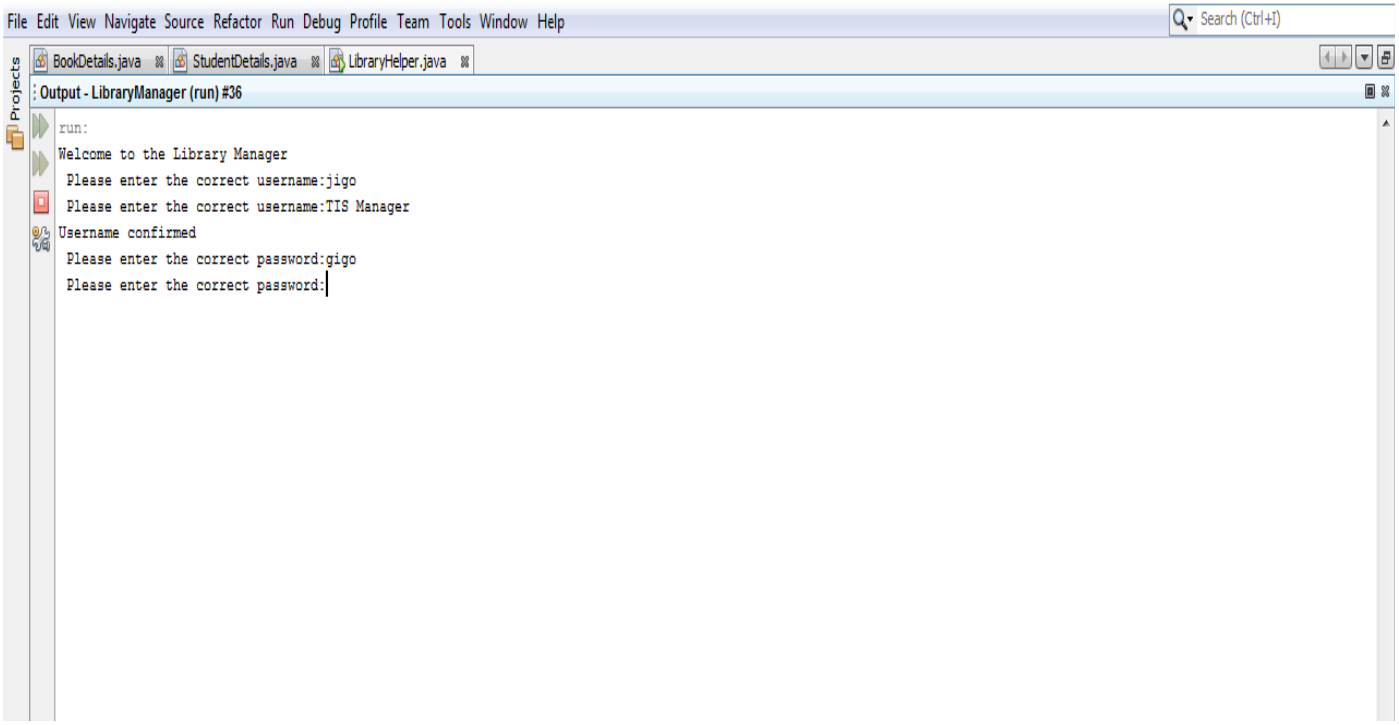
When the user inputs the wrong username the program asks the asks the user to input the correct username again:



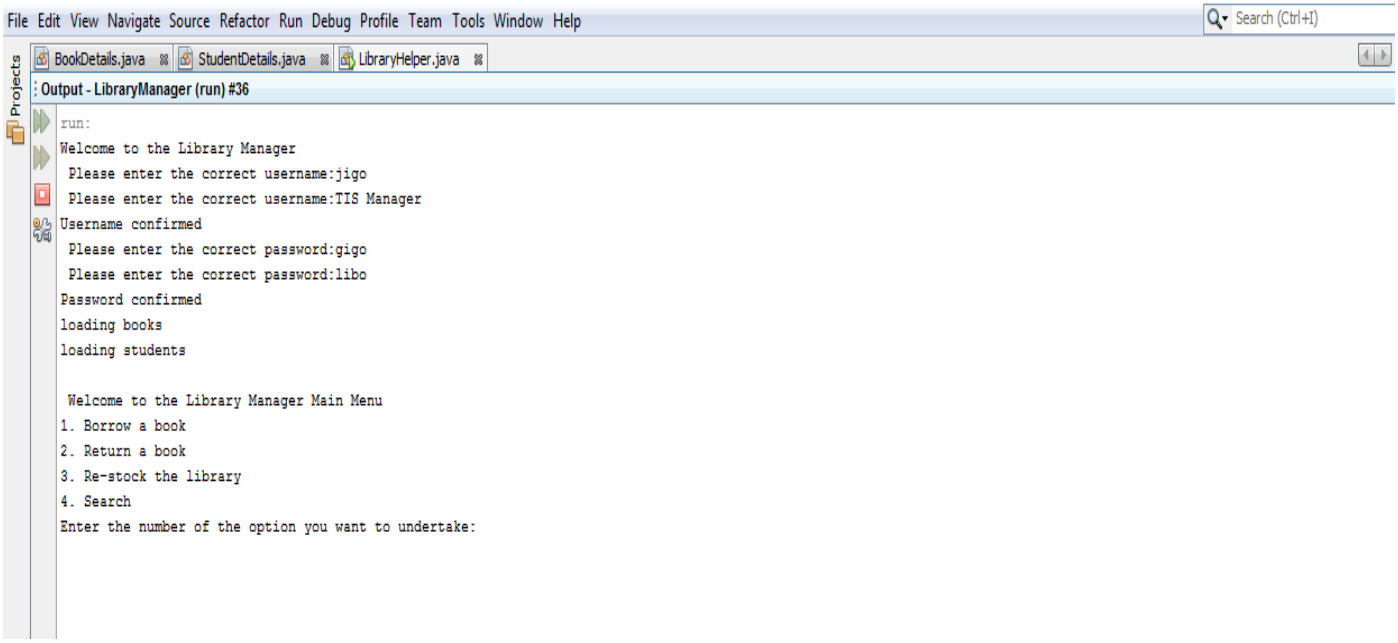
However, when the correct username is input by the user, the program goes on to ask the user to input the correct password:



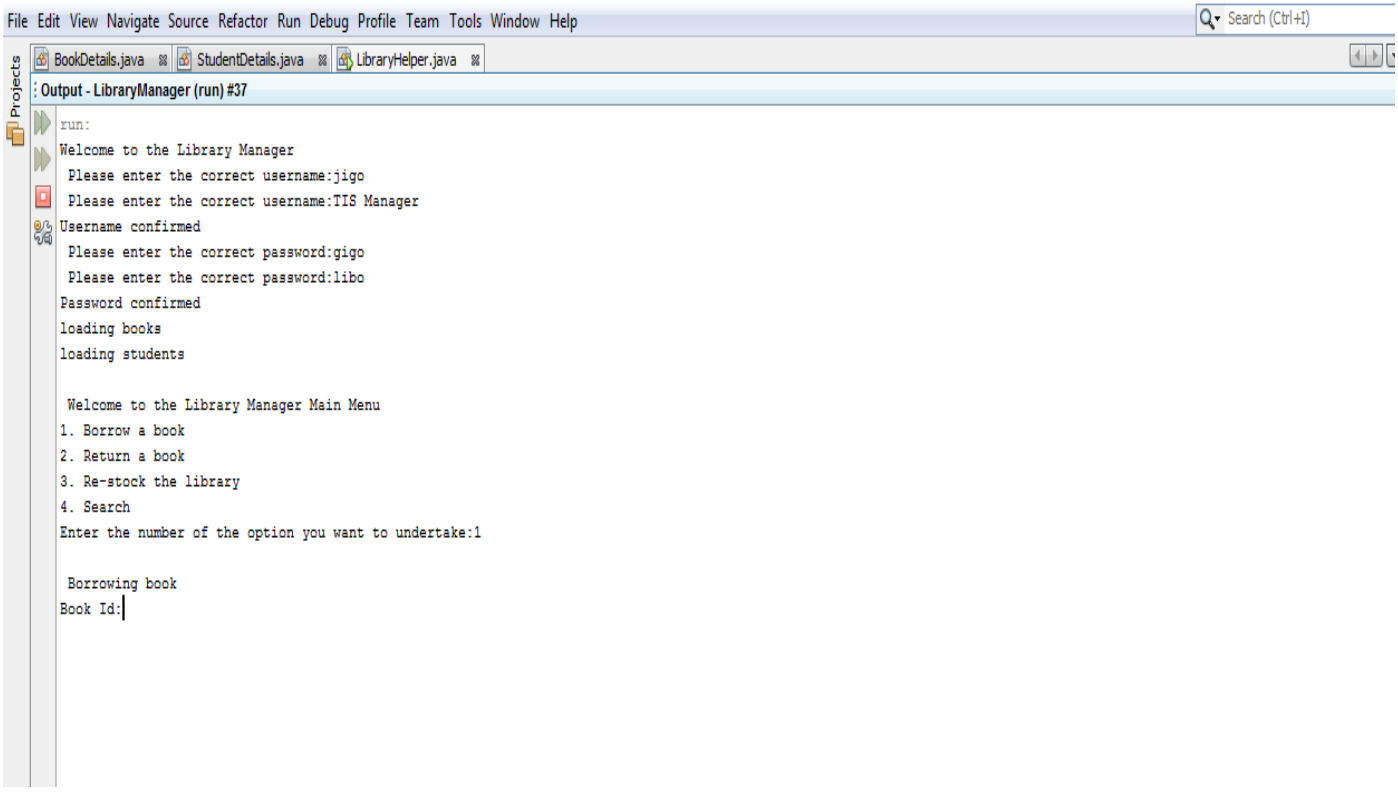
Just like the username, the program keeps asking the user to input the correct password if the wrong password is inputted:



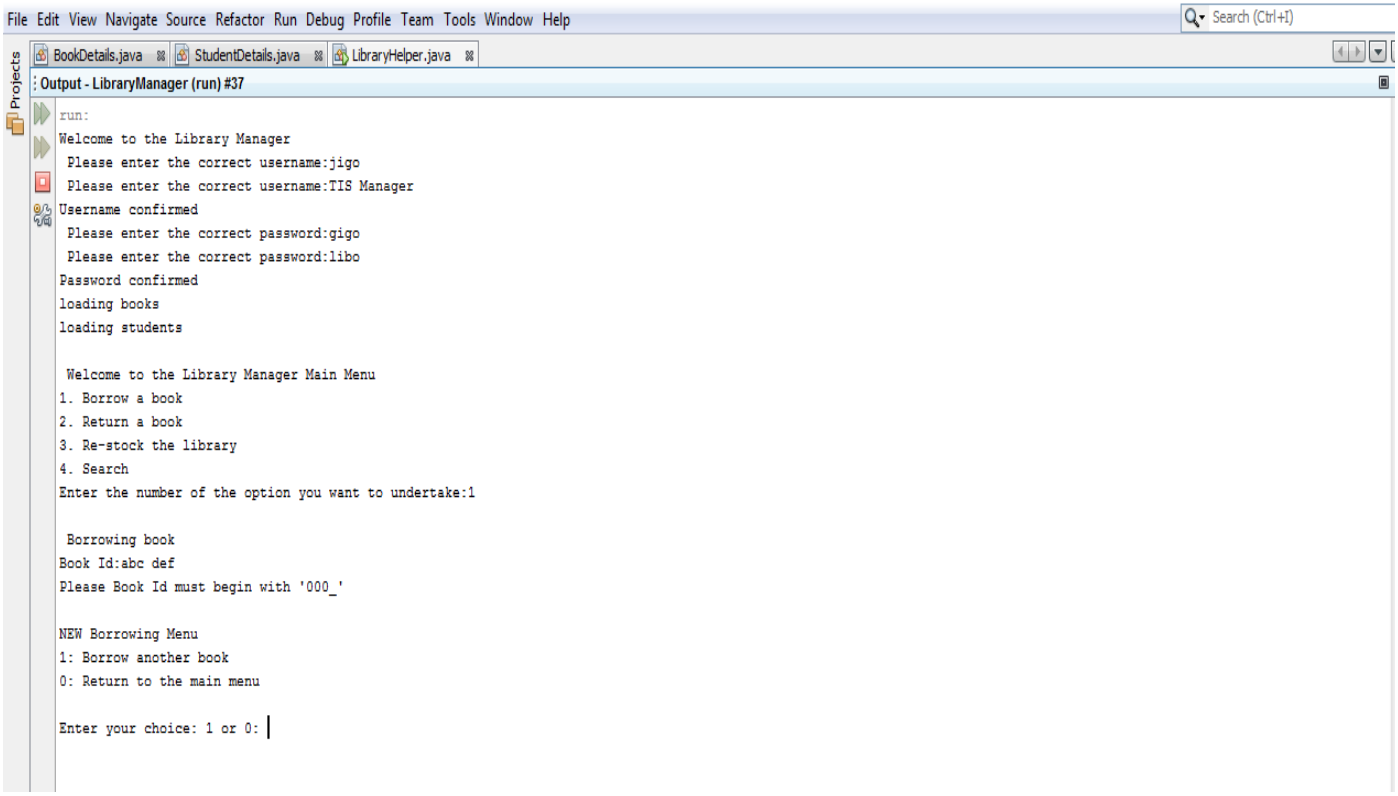
However, when the correct password is entered the program then loads the student and book profiles; and, then it goes on to show the user the main menu:



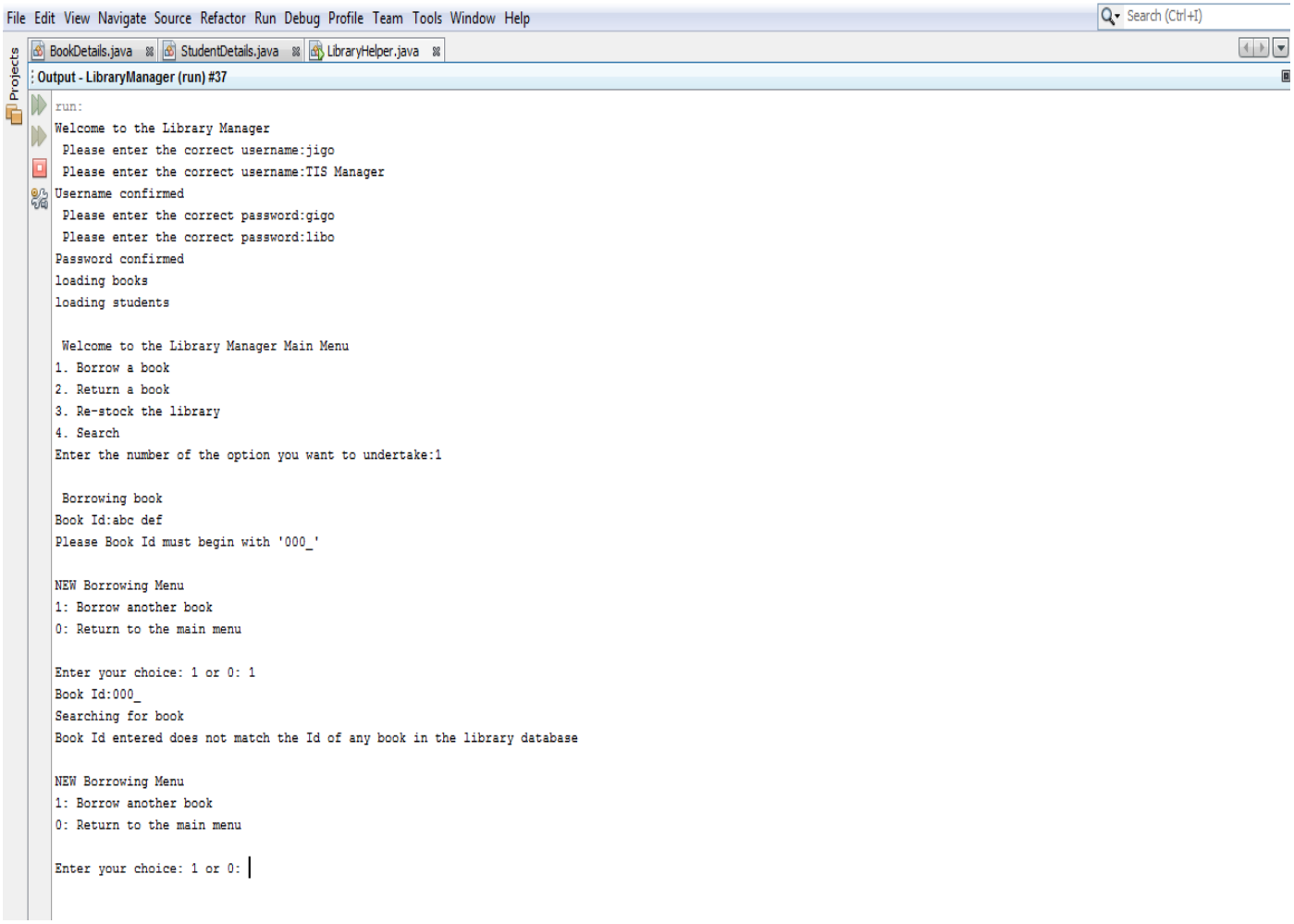
At this stage, the user has to input the number of the options listed in the main menu; thus if the user decides to allow a student to borrow a book, then he/she input the number 1 into the program. And after this number is inputted the user is directed to the borrowing interface:



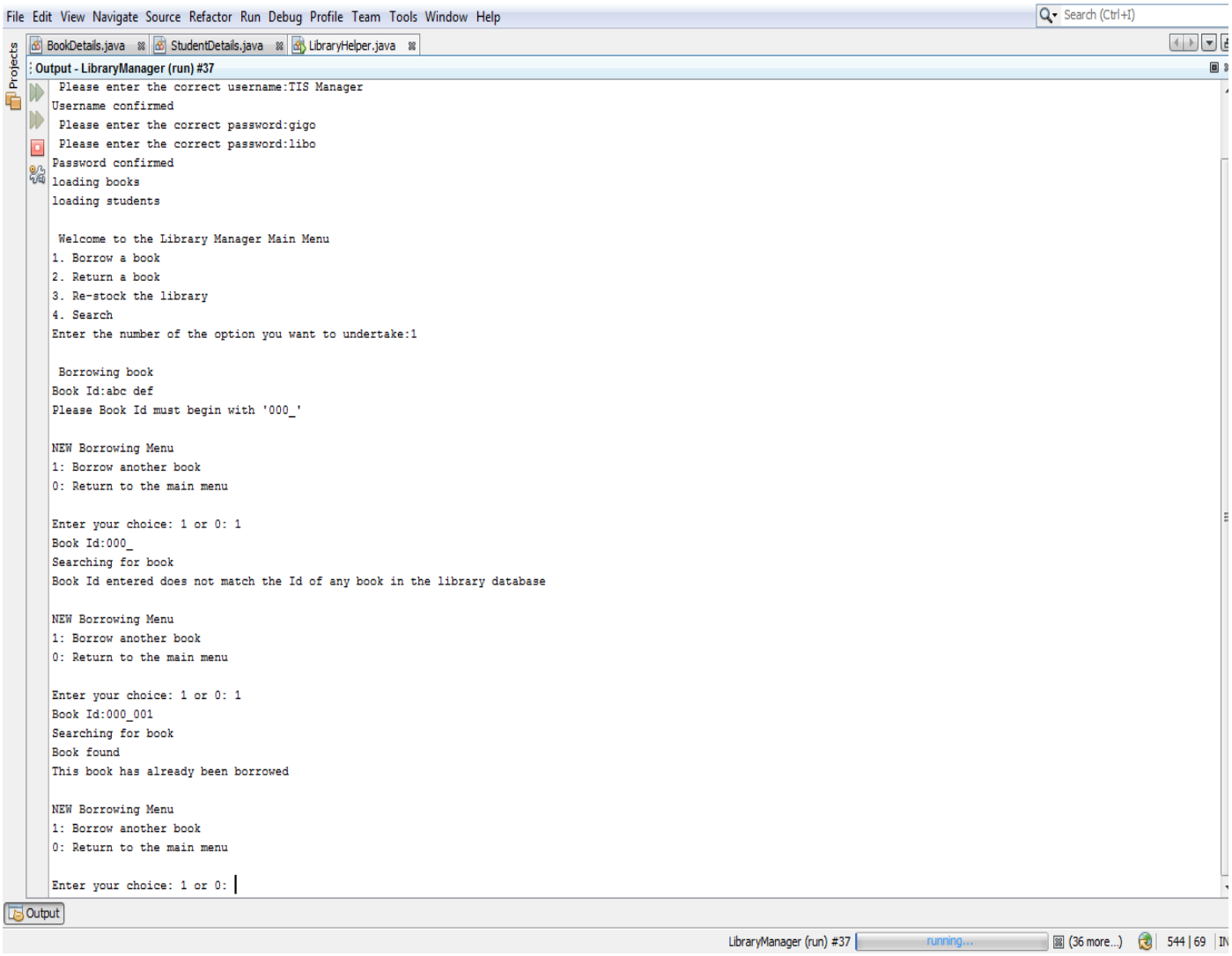
At this stage user will not be allowed to input any data apart from data input that begins with ‘000_’; thus when the user inputs the wrong data, the following screen appears:



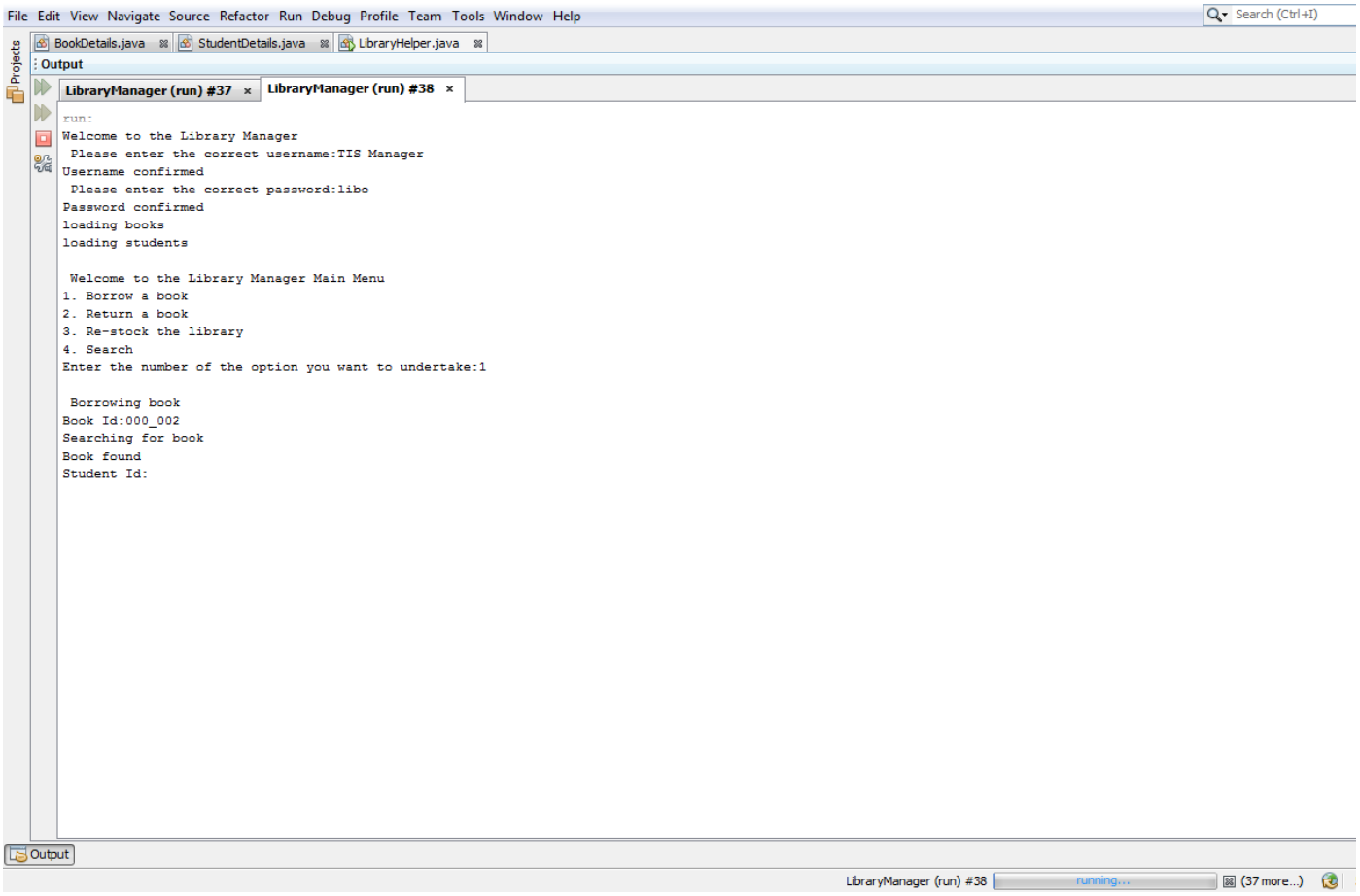
From here, the only way the user can continue to borrow the book is if the user input 1 into the new borrowing menu. So, when the user inputs 1 into the new borrowing menu, the user is then allowed to input the book Id of the book to be borrowed. If the user inputs an Id that begins with ‘000_’ the program then goes on to check whether that book Id exists; so even if the book Id entered may begin with ‘000_’, if an invalid book Id is entered the program will tell the user that the book Id does not exist:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
BookDetails.java
StudentDetails.java
LibraryHelper.java
Output - LibraryManager (run) #37
run:
Welcome to the Library Manager
Please enter the correct username:jigo
Please enter the correct username:TIS Manager
Username confirmed
Please enter the correct password:gigo
Please enter the correct password:libo
Password confirmed
loading books
loading students
Welcome to the Library Manager Main Menu
1. Borrow a book
2. Return a book
3. Re-stock the library
4. Search
Enter the number of the option you want to undertake:1
Borrowing book
Book Id:abc def
Please Book Id must begin with '000_'
NEW Borrowing Menu
1: Borrow another book
0: Return to the main menu
Enter your choice: 1 or 0: 1
Book Id:000_
Searching for book
Book Id entered does not match the Id of any book in the library database
NEW Borrowing Menu
1: Borrow another book
0: Return to the main menu
Enter your choice: 1 or 0:

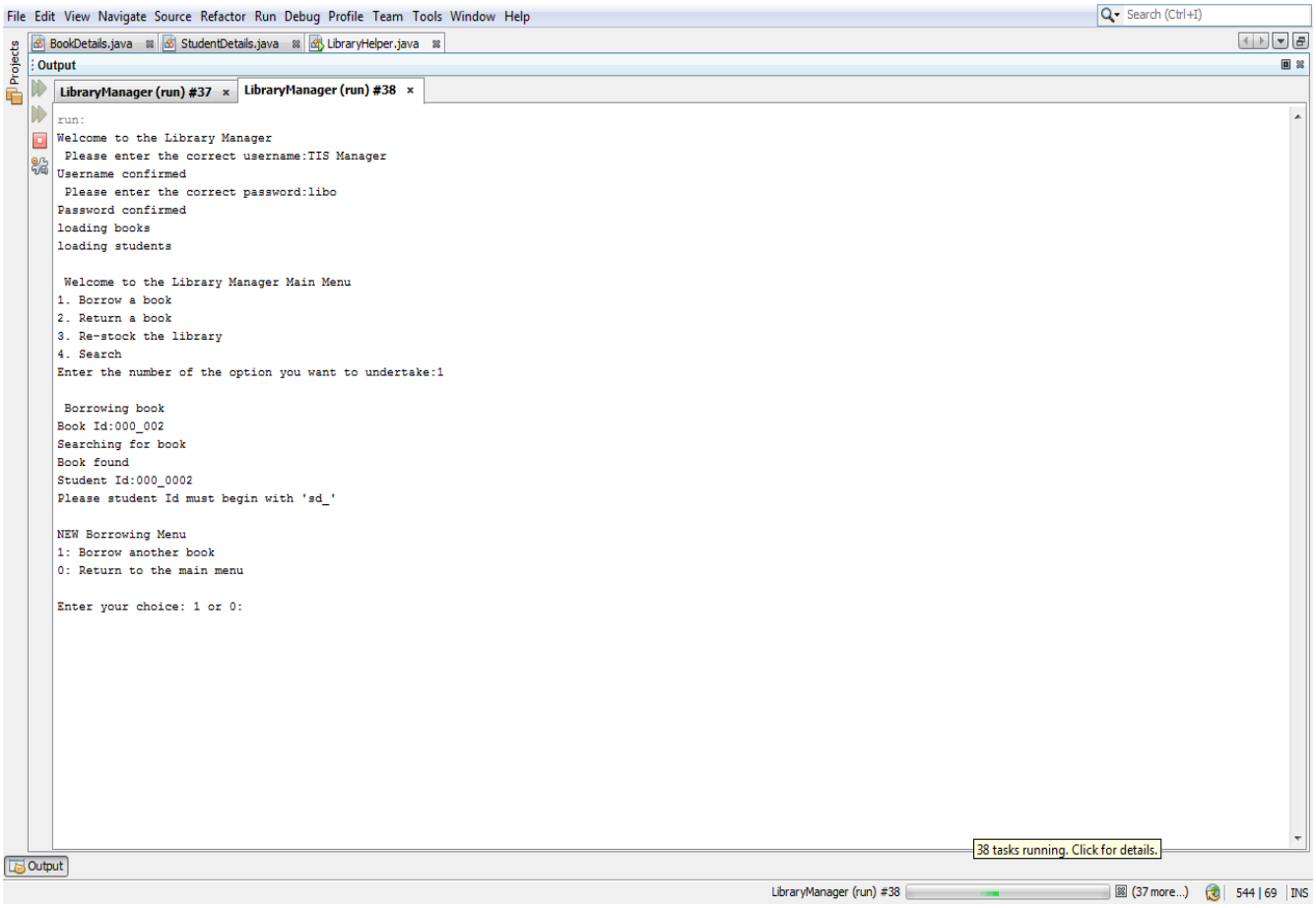
Another validation check at this stage if the inputted book has already been borrowed; the screenshot below shows the message displayed to the user when the program discovers this:



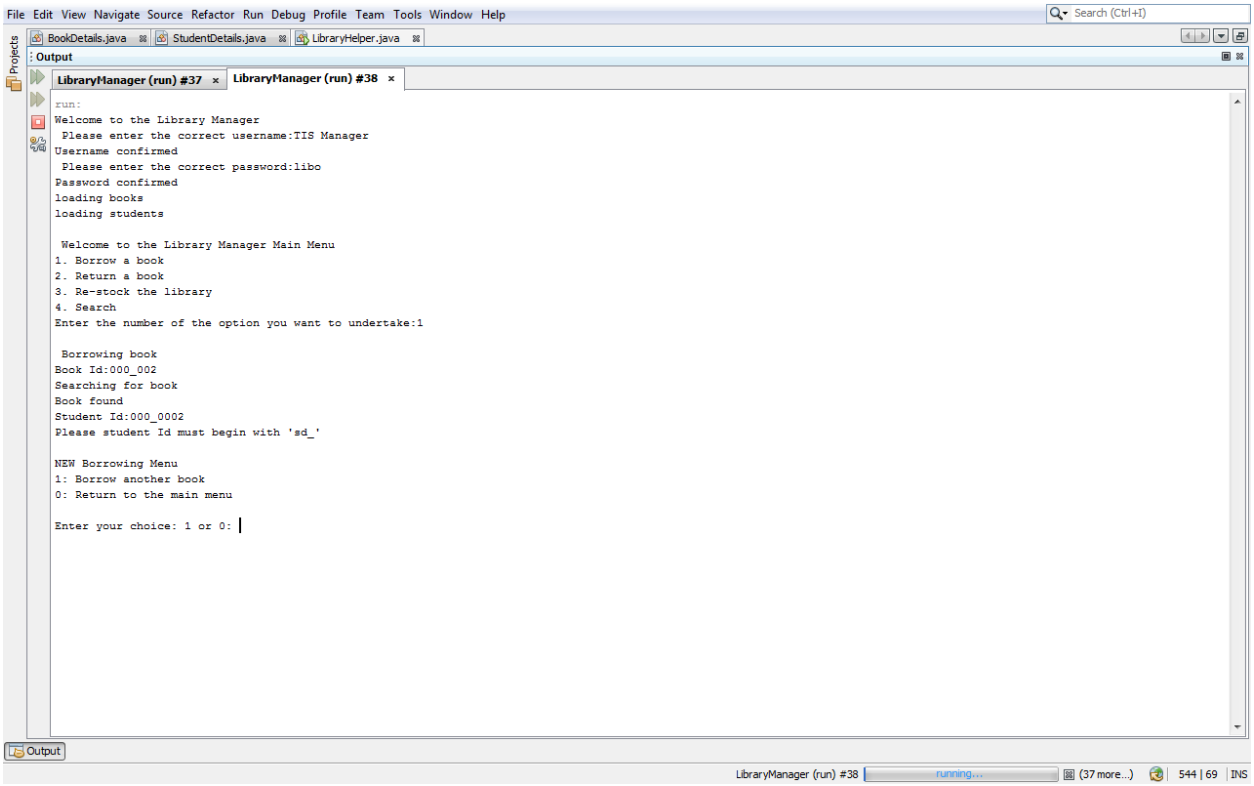
Thus, the user will only be allowed to enter a book that has not been borrowed; and, when this is entered, the program goes on to ask the user to input the Id of the student borrowing the book:



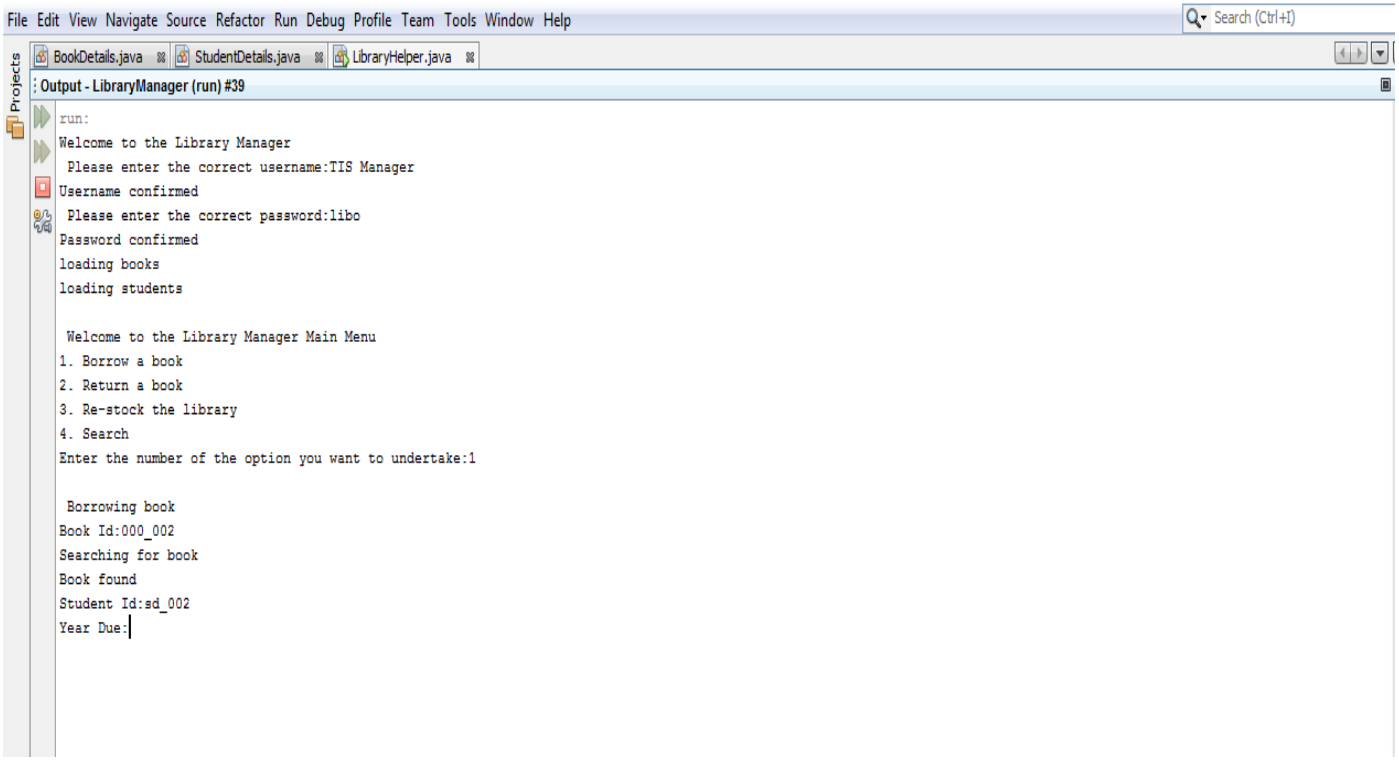
Also at this stage when the user inputs a student Id that does not begin with ‘sd_’, the program displays the error message:



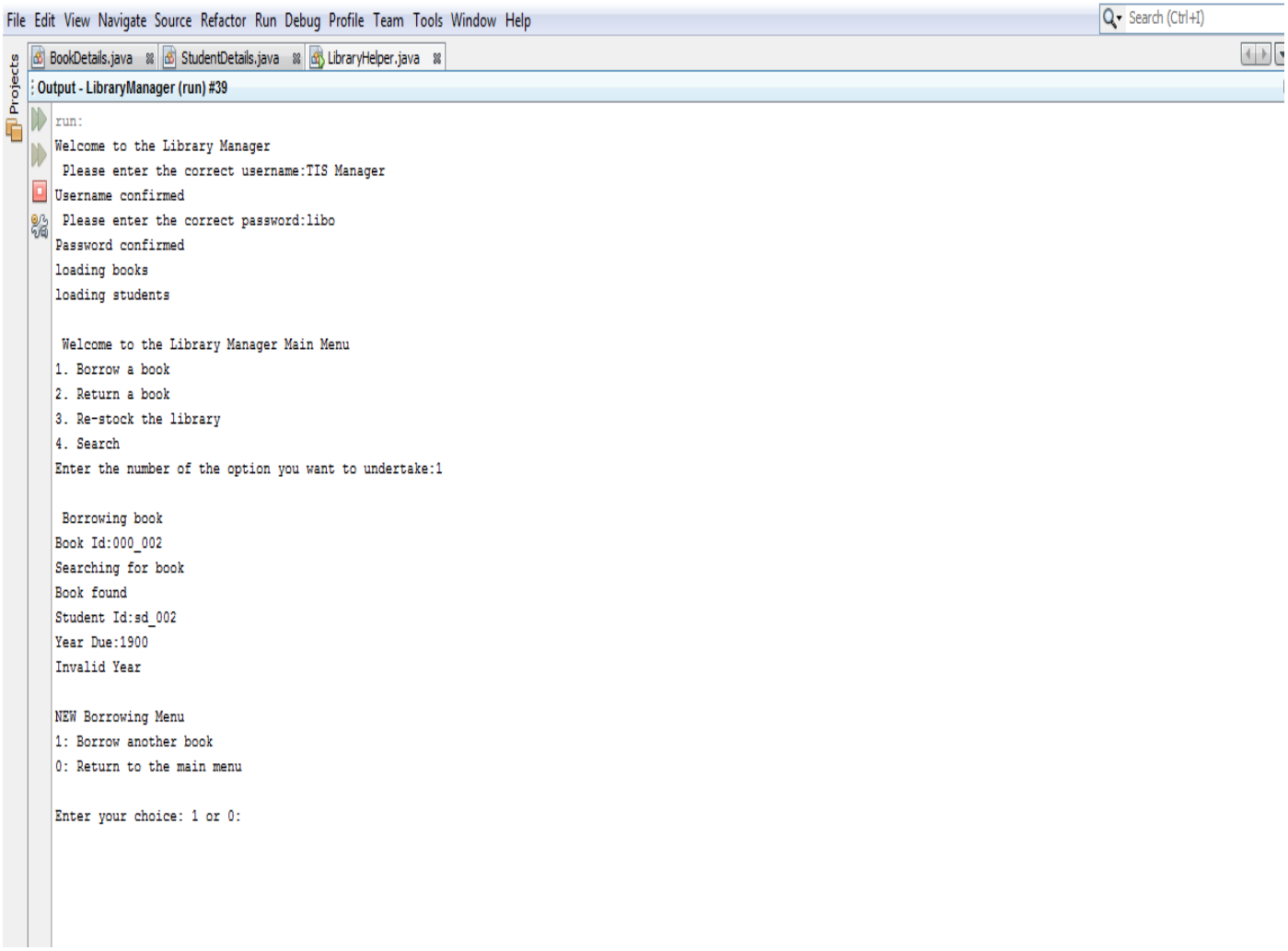
Also when the user inputs a student who already has a book borrowed, the program displays an error message to inform the user of this:

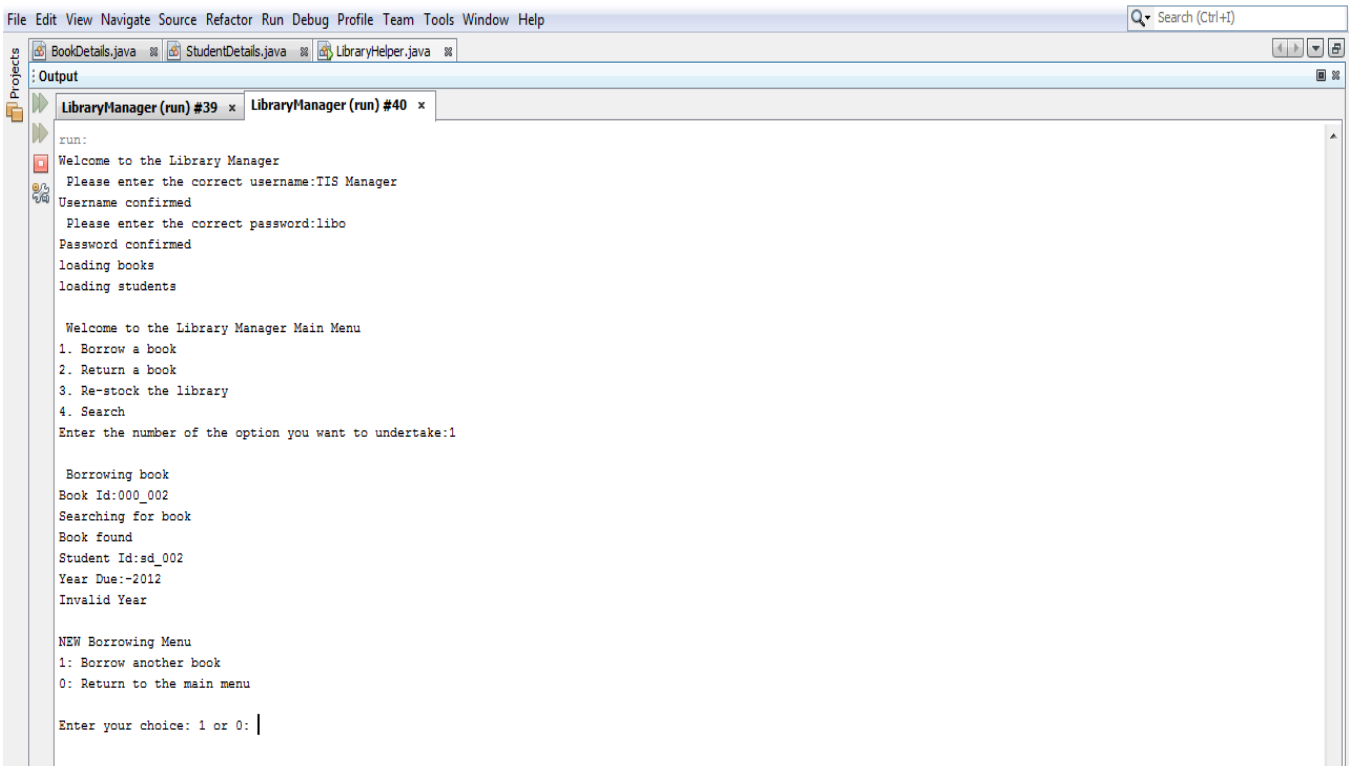


The only way the user will be allowed to input the other details of the student borrowing the book is if the correct student Id is entered, and if the student entered does not have any book in his or her possession; the screenshot of this is shown below:

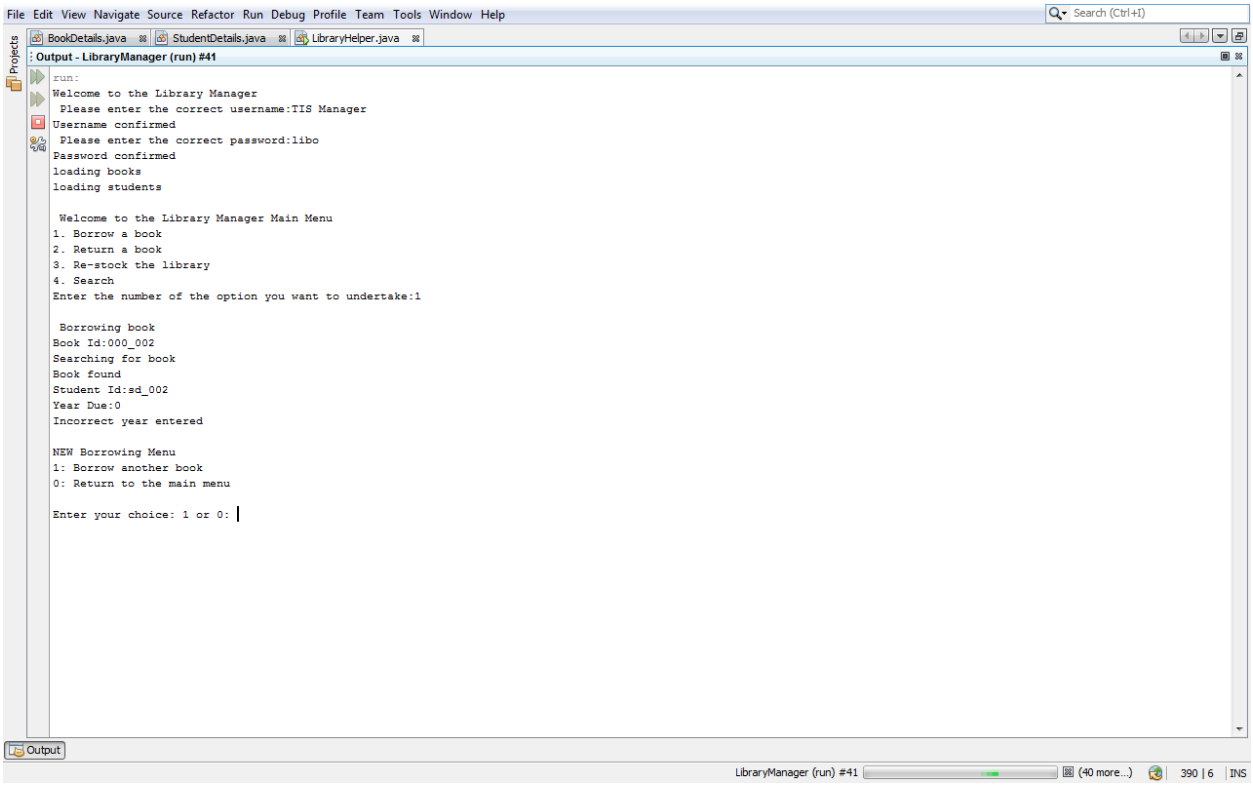


After the user inputs the correct input for the book Id, and student Id during the borrowing stage, the program then asks the user to input the year the borrowed book should be due. At this stage the user can only input years from 2012, thus any other input will let the program display an error message:

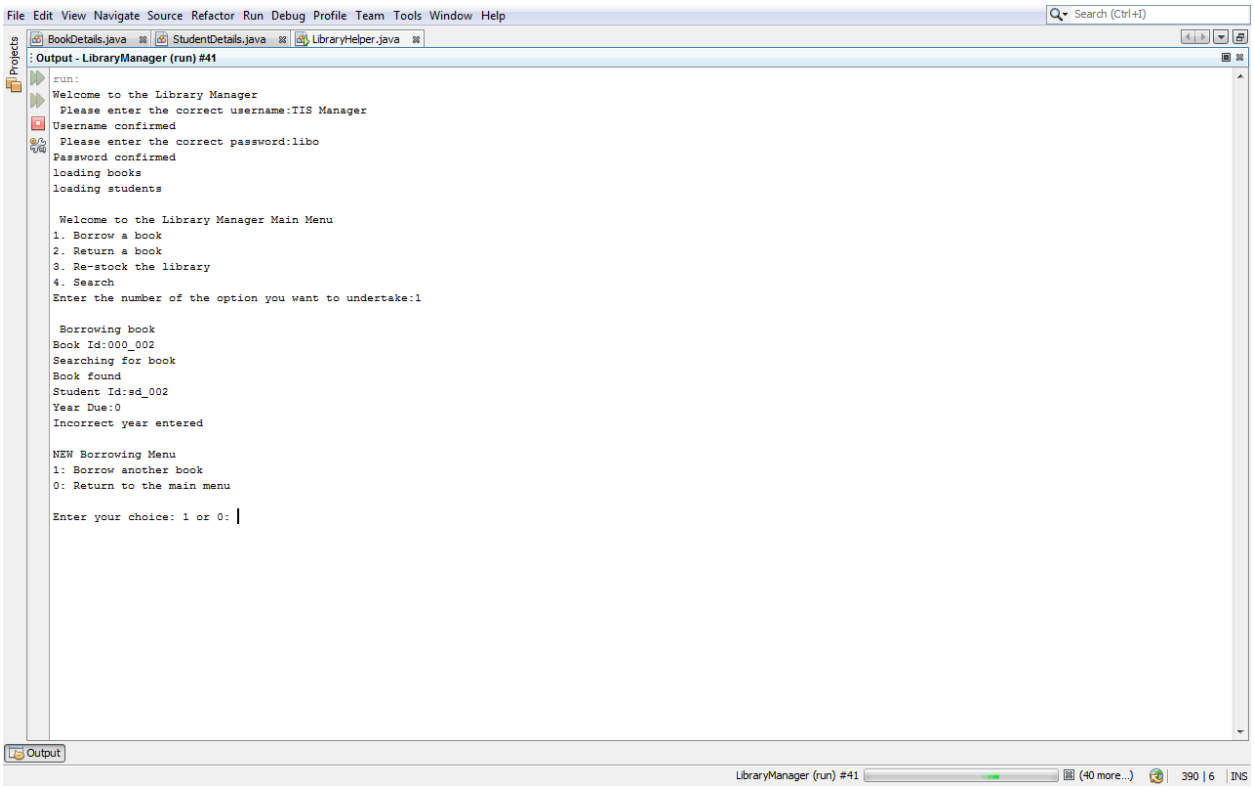




The above is input for a negative year

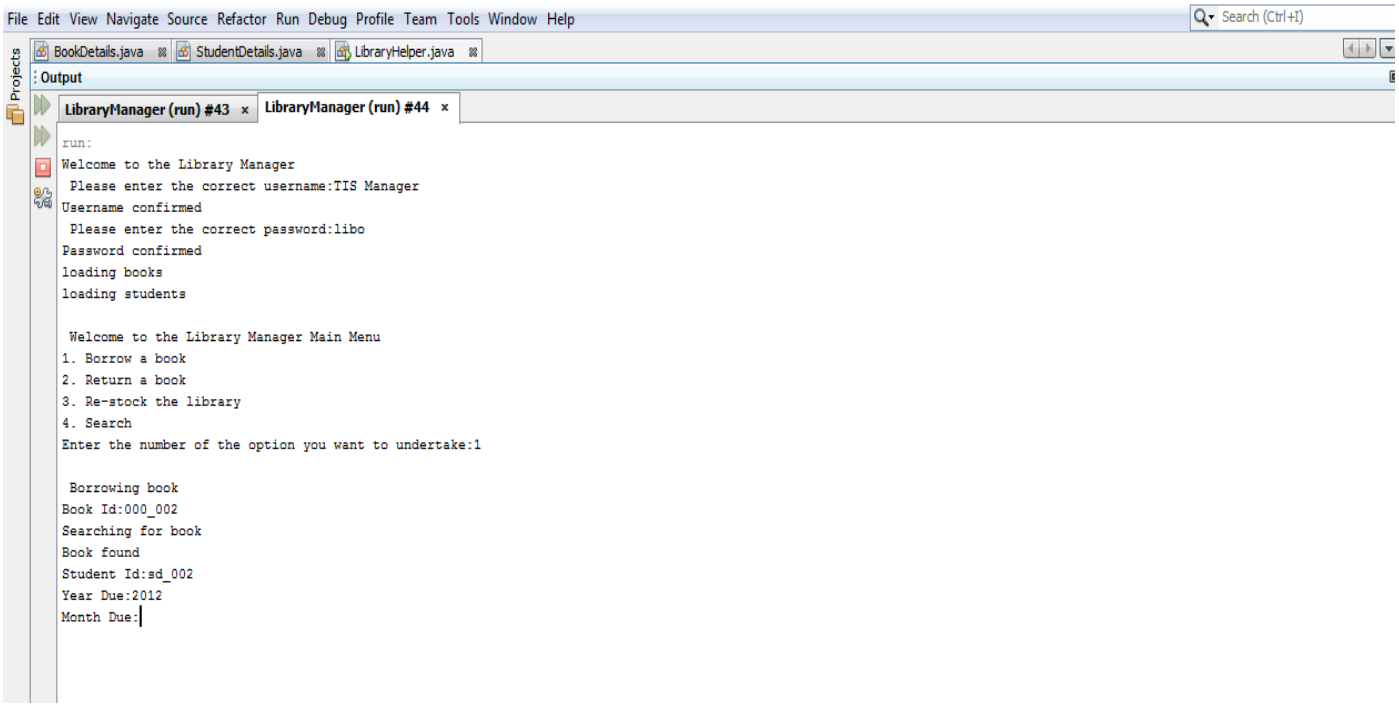


The above is input for when zero is entered

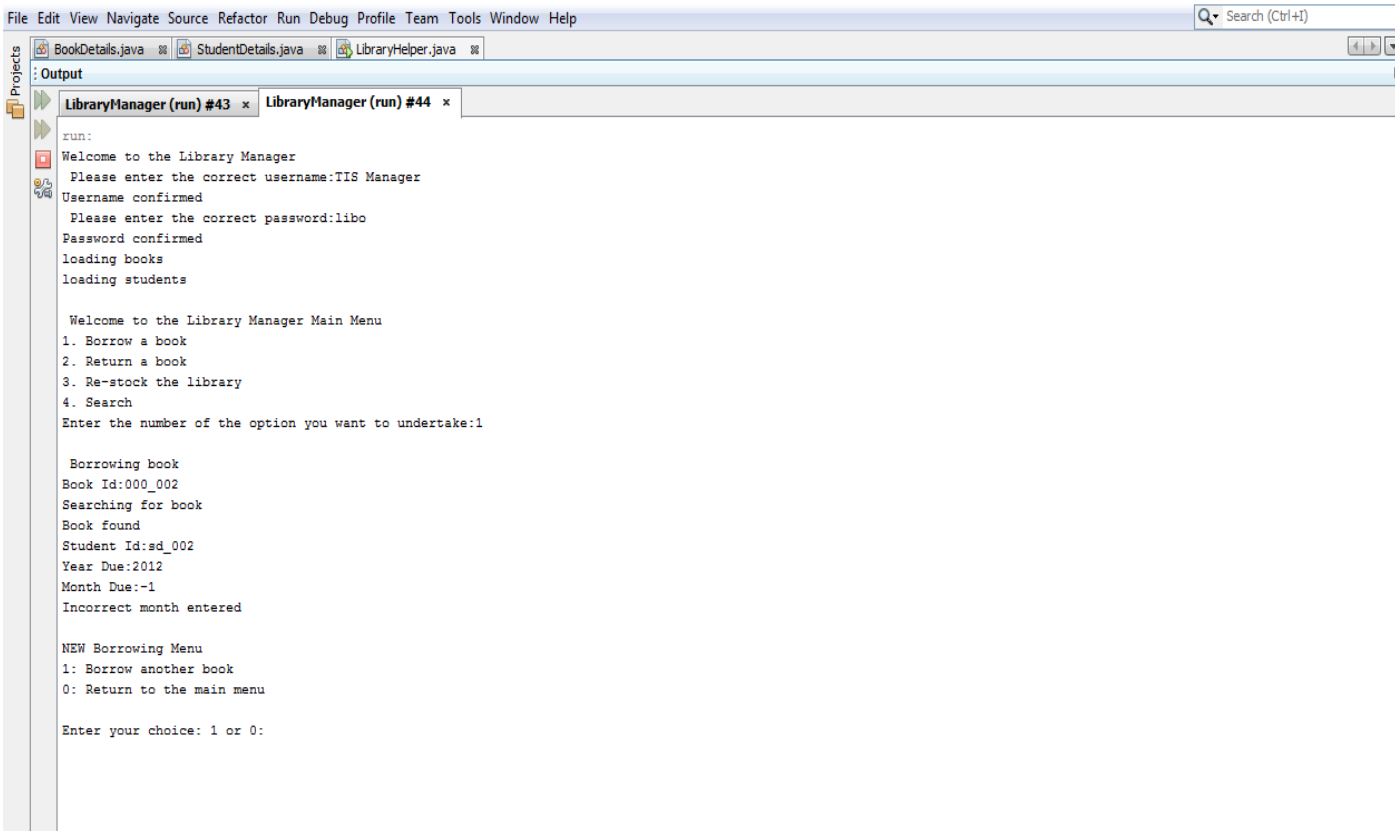


The above is a diagram for when a string is inputted in this field

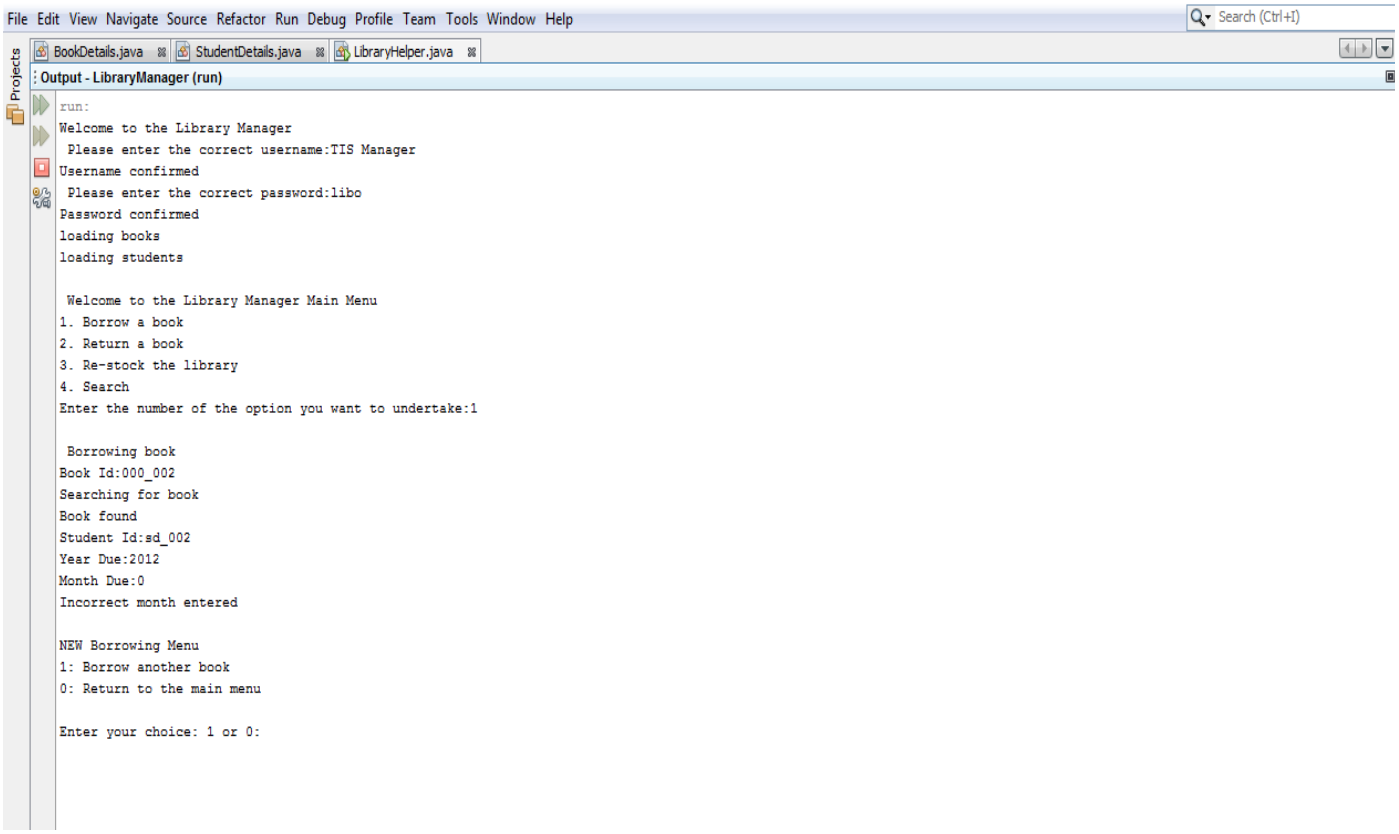
Thus only when the correct input for year is entered, will the program allow the user to input the month and day the borrowed book is due:



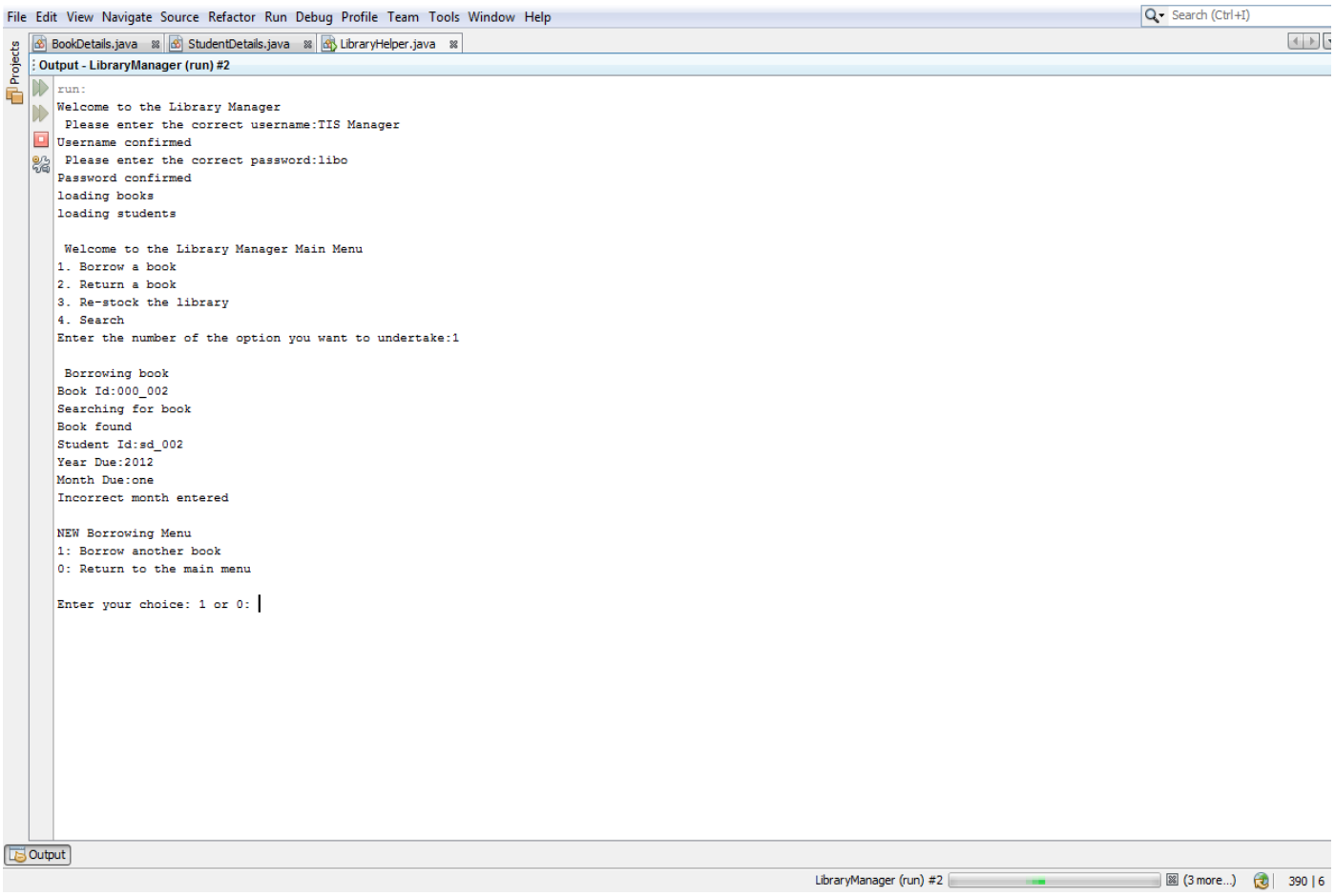
At this stage, the correct input for the month will be any integer from 1 to 12, any other input apart from this will display the following error message:



The above is for when a negative number is inputted

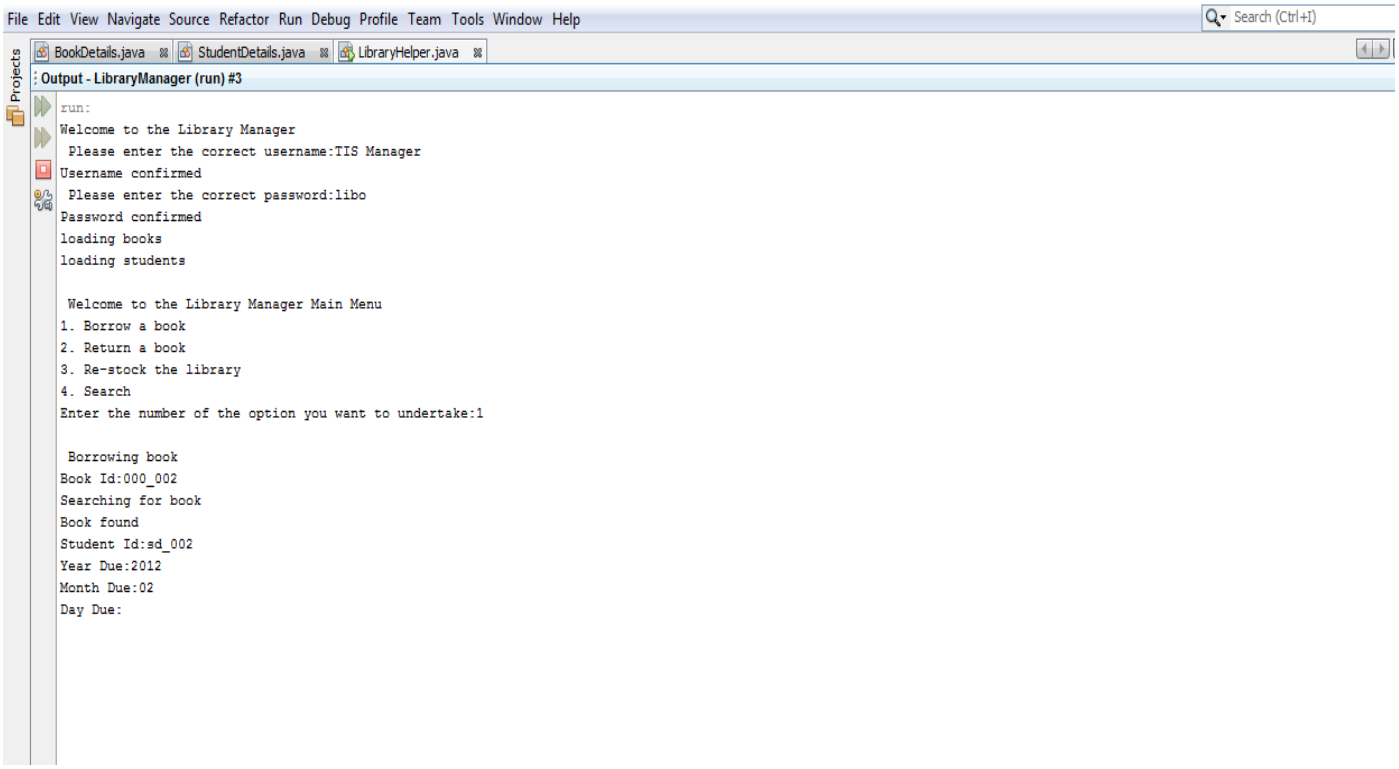


The above is for when zero is inputted

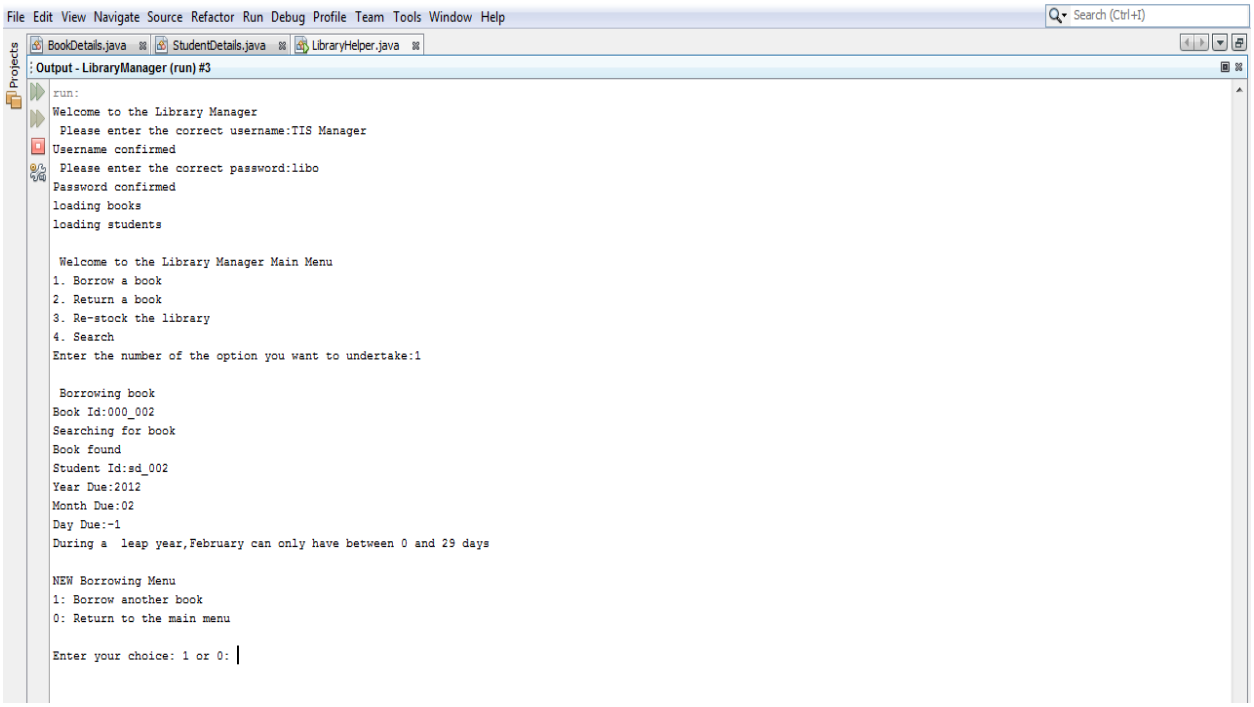


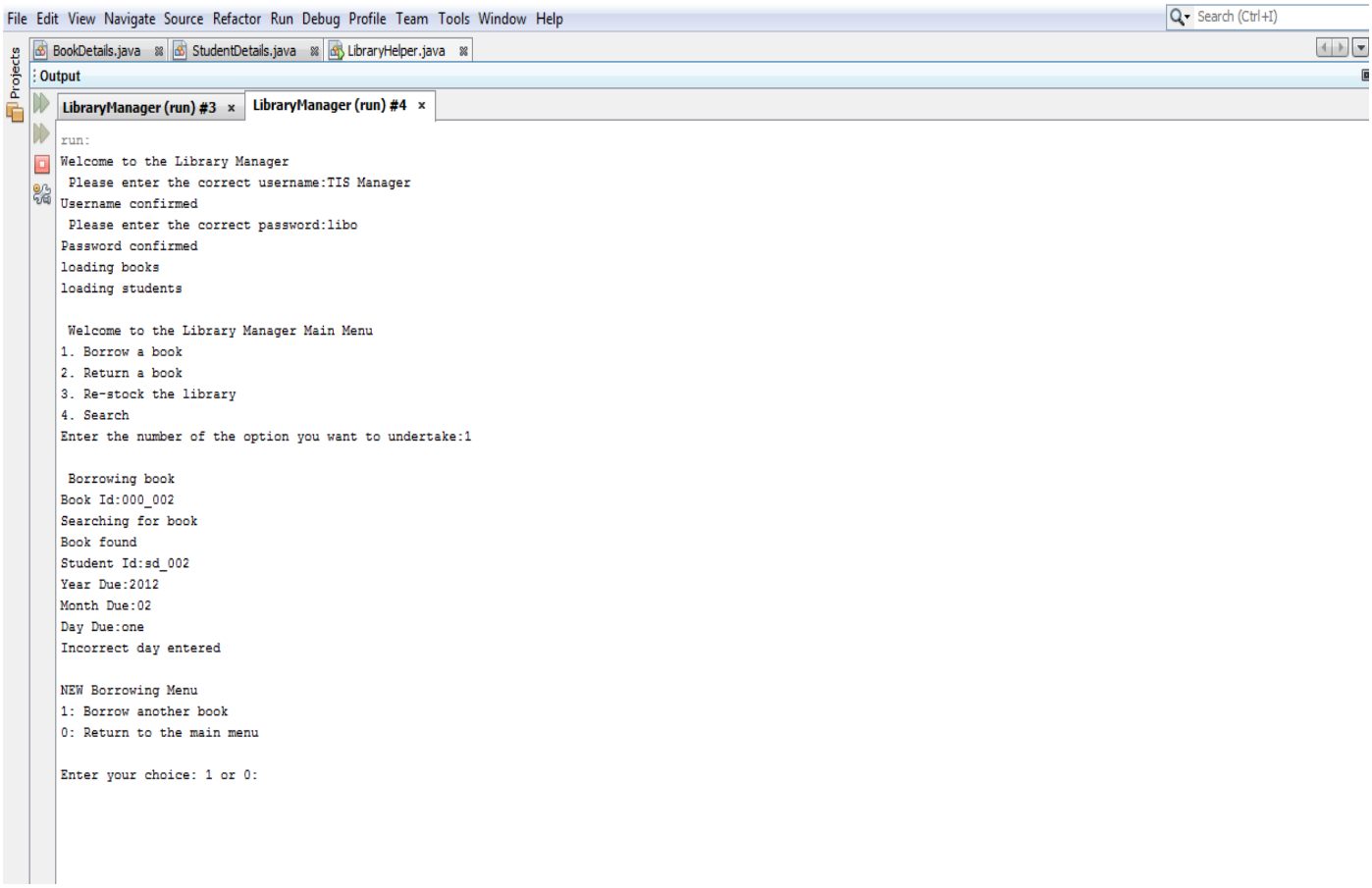
The above is for when a string is inputted

After the correct input for the month is inputted, the program asks the user to input the day the borrowed book should be due:

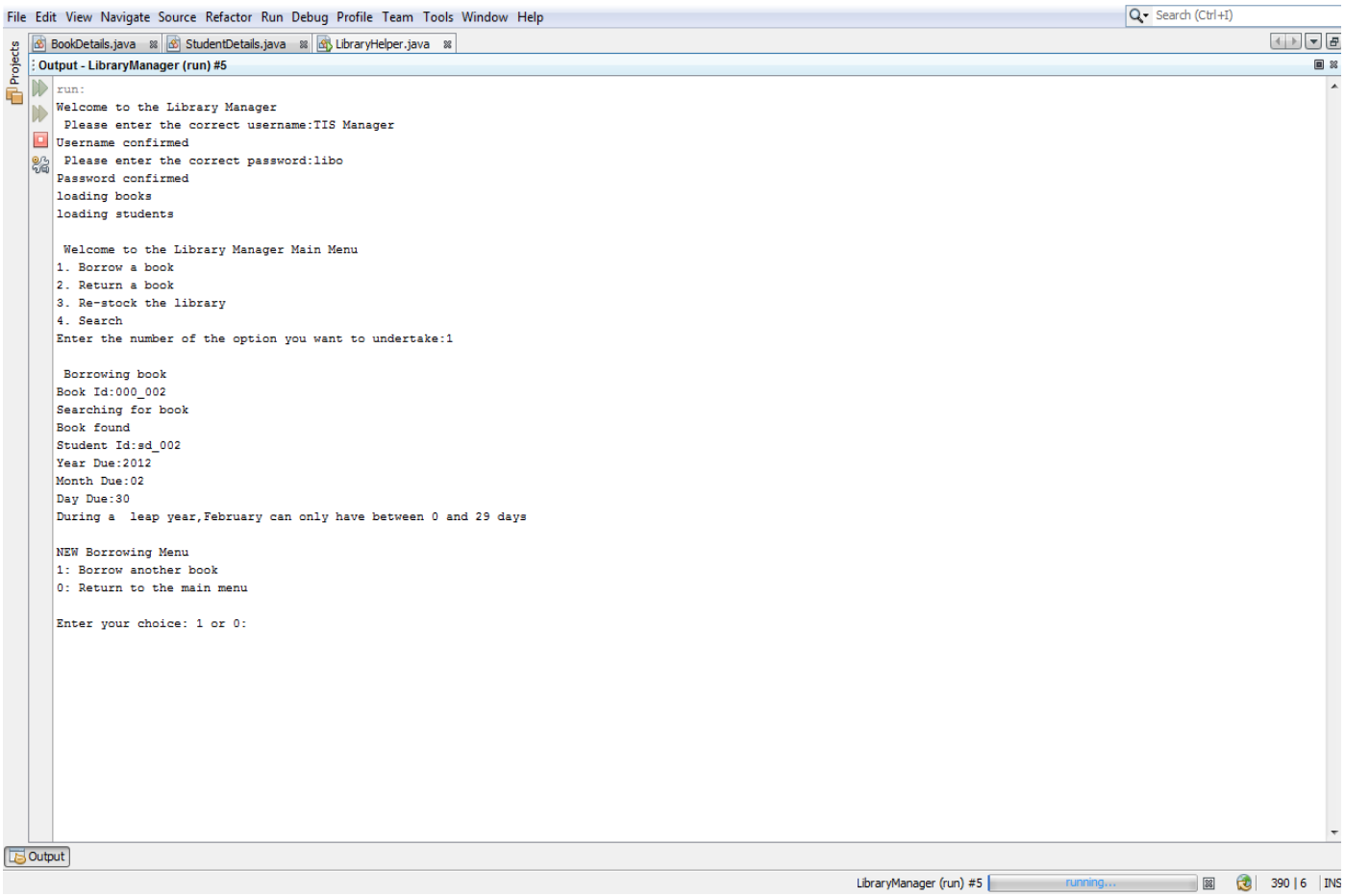


Also at the stage the day must be entered, the user cannot input string and negative numbers- the displays error messages when these are entered. The screenshot for the stage invalid data is entered at this stage is:



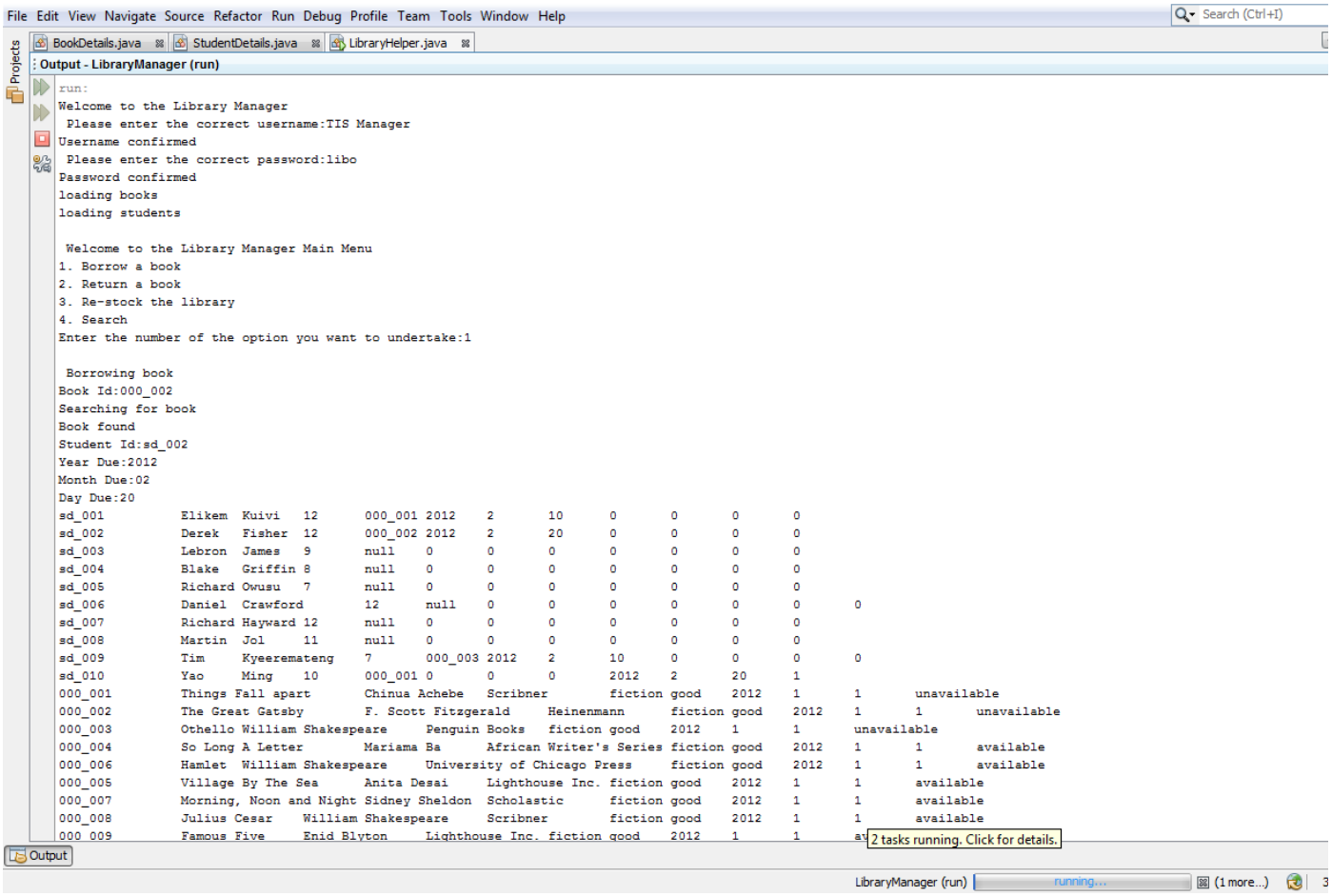


The above is for when a string is entered in this field.



The above is input when the user inputs more than 29 days for the month of February

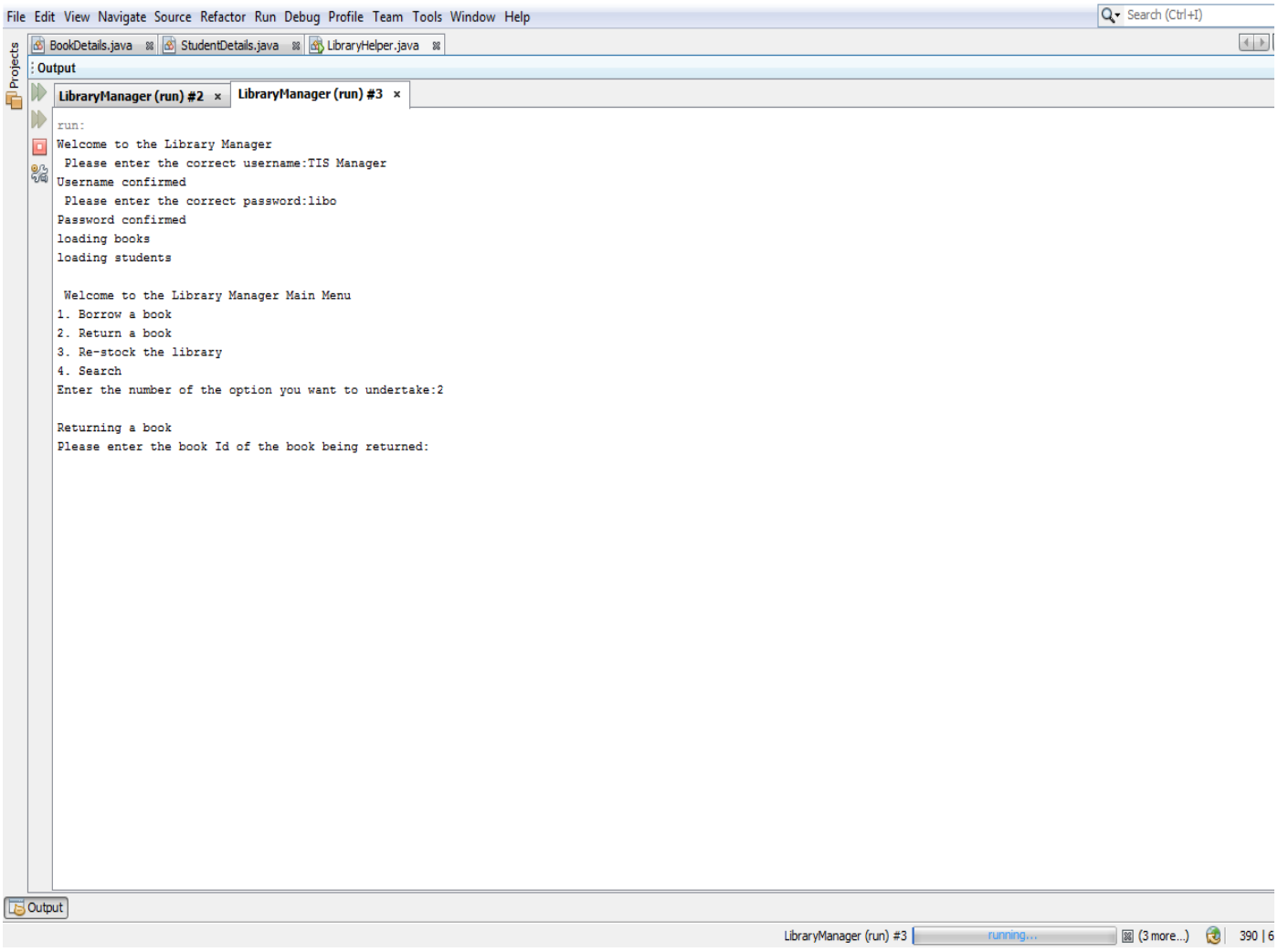
When the user inputs the correct data for the day, then the program saves all the changes to the student and book profiles to show that the respective books have been borrowed. The screenshots below show this:



The diagram above shows that the student with the Id ‘sd_002’ has borrowed a book with the Id ‘000_002’ and that the book is due on the 20th of February 2012. Also the status of the borrowed book becomes ‘unavailable’.

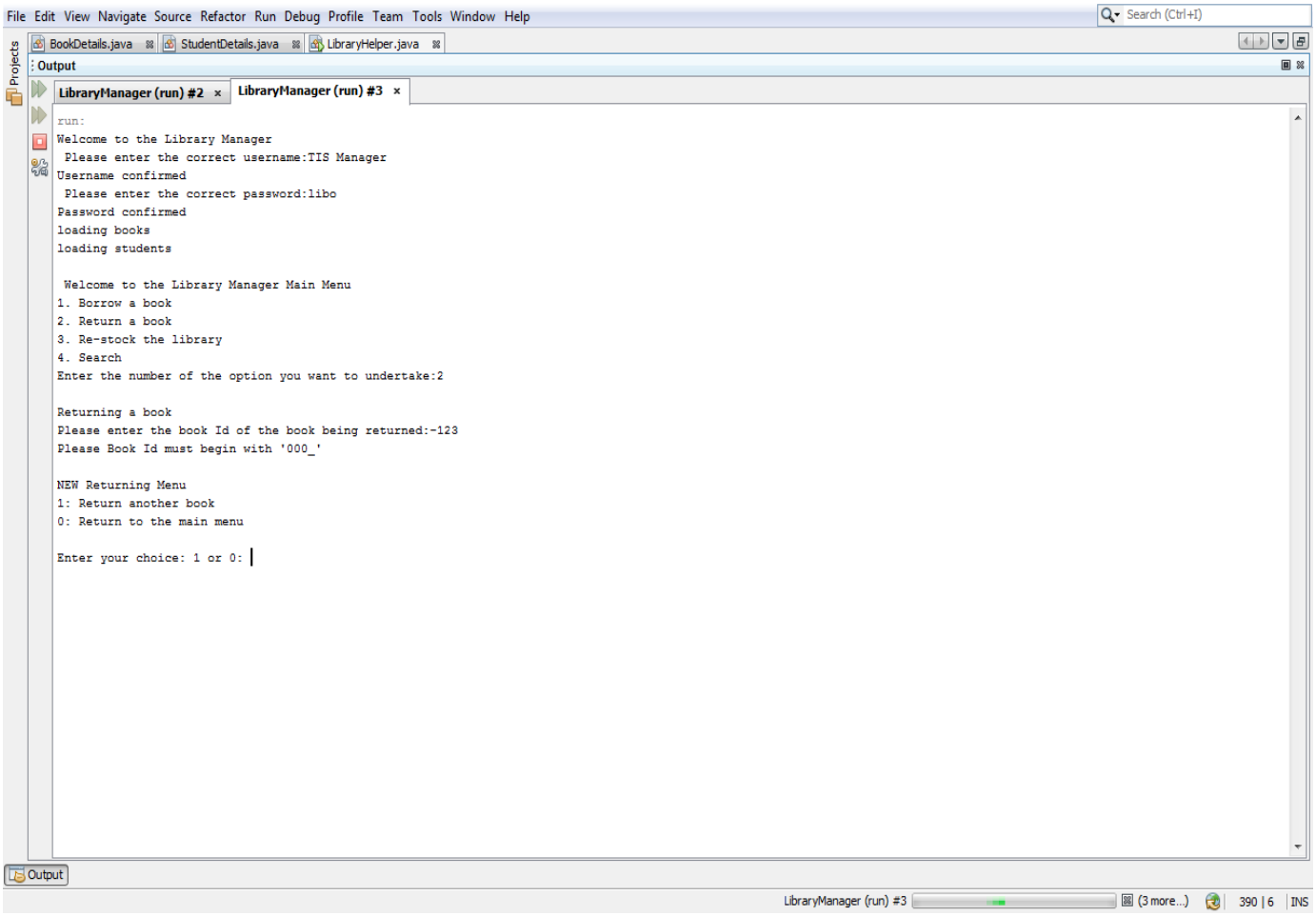
After the book has been borrowed, the program then gives the user to borrow out another book, or to return to the main menu.

Now when the user enters the returning interface, the first thing the program asks the user to do is to enter the Id of the book being returned:

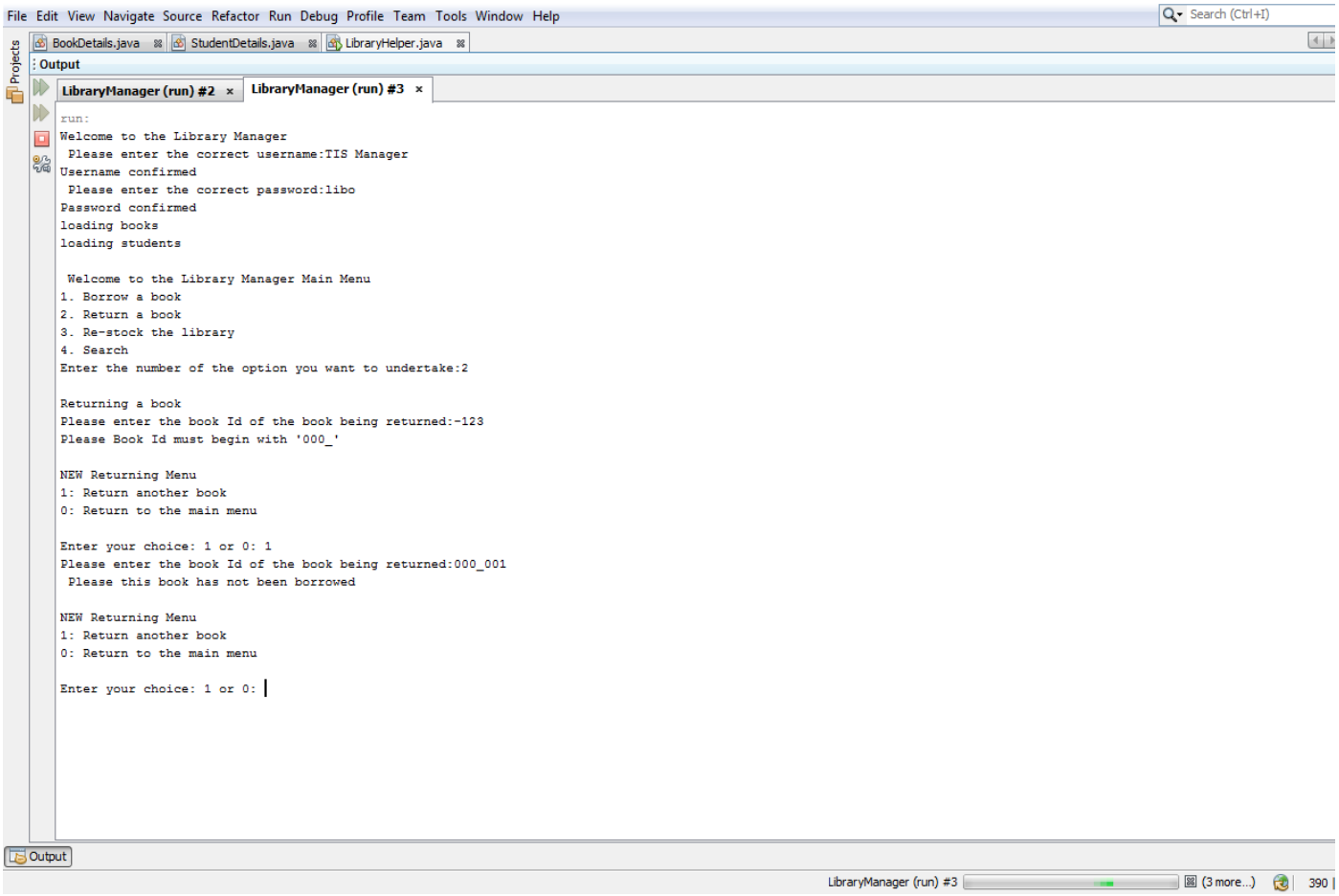


At this stage also, the user can only enter a book Id that begins with ‘000_’, any other input will cause the program to display an error message:

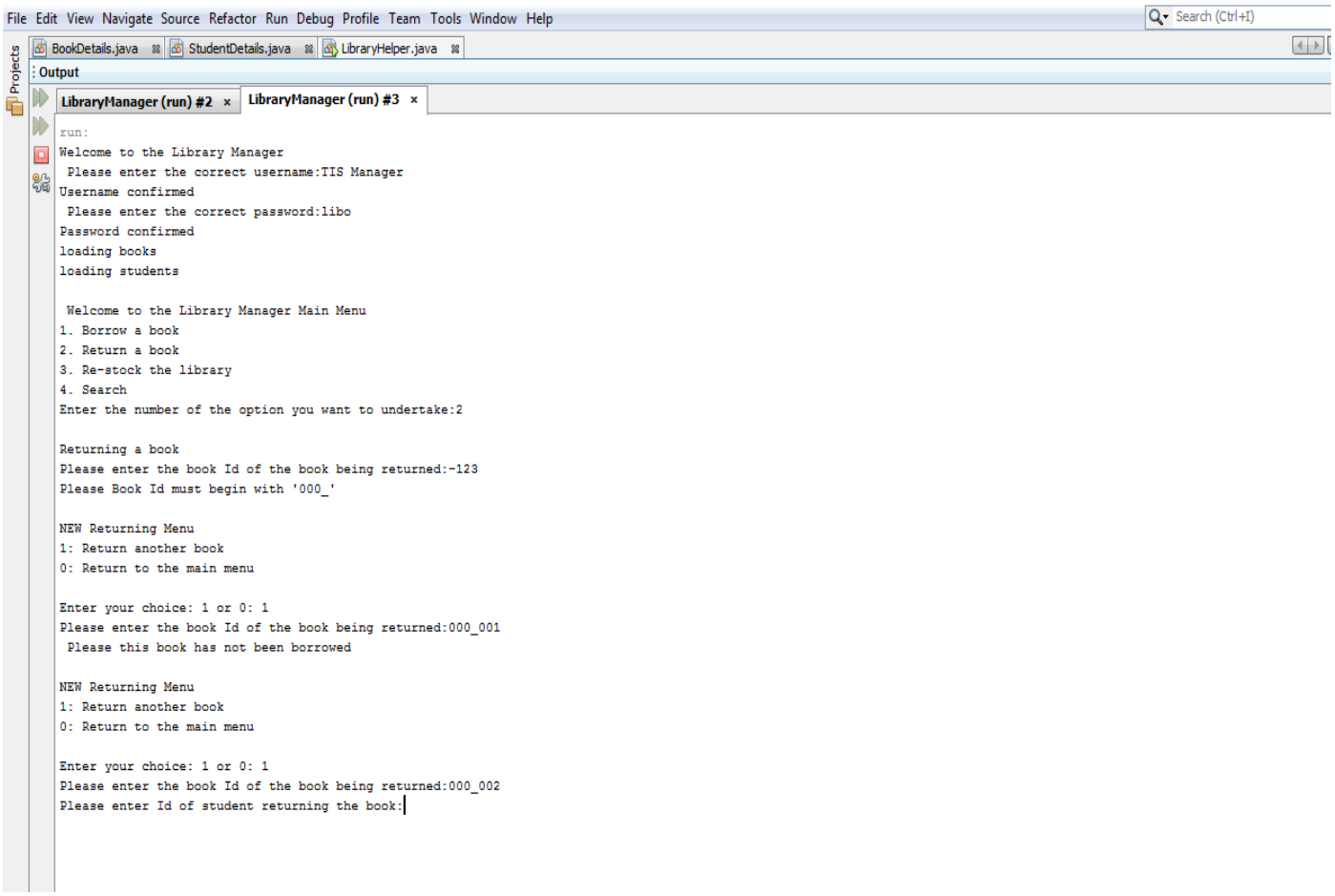
a



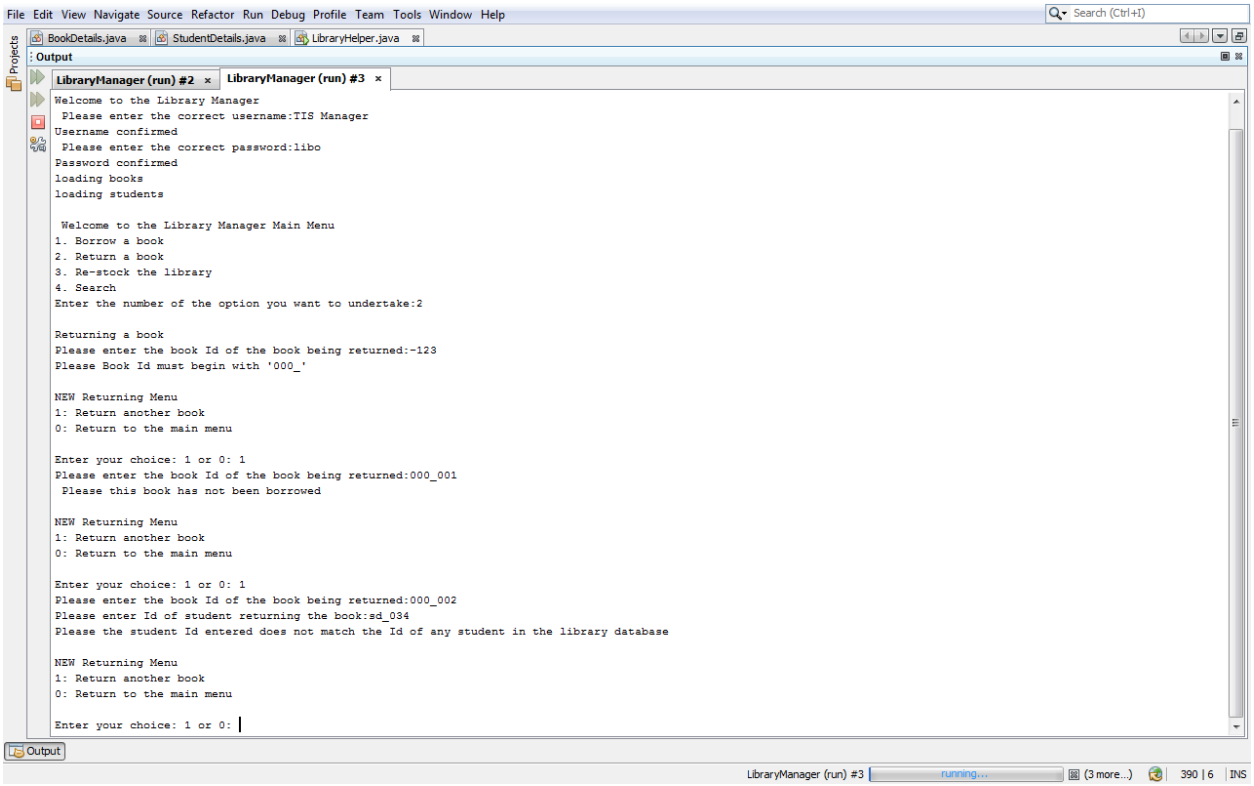
Also, at this stage, the user must input the Id of a book that has been borrowed; if the book inputted at this stage has not been borrowed, the program will display an error message to inform the user of this:



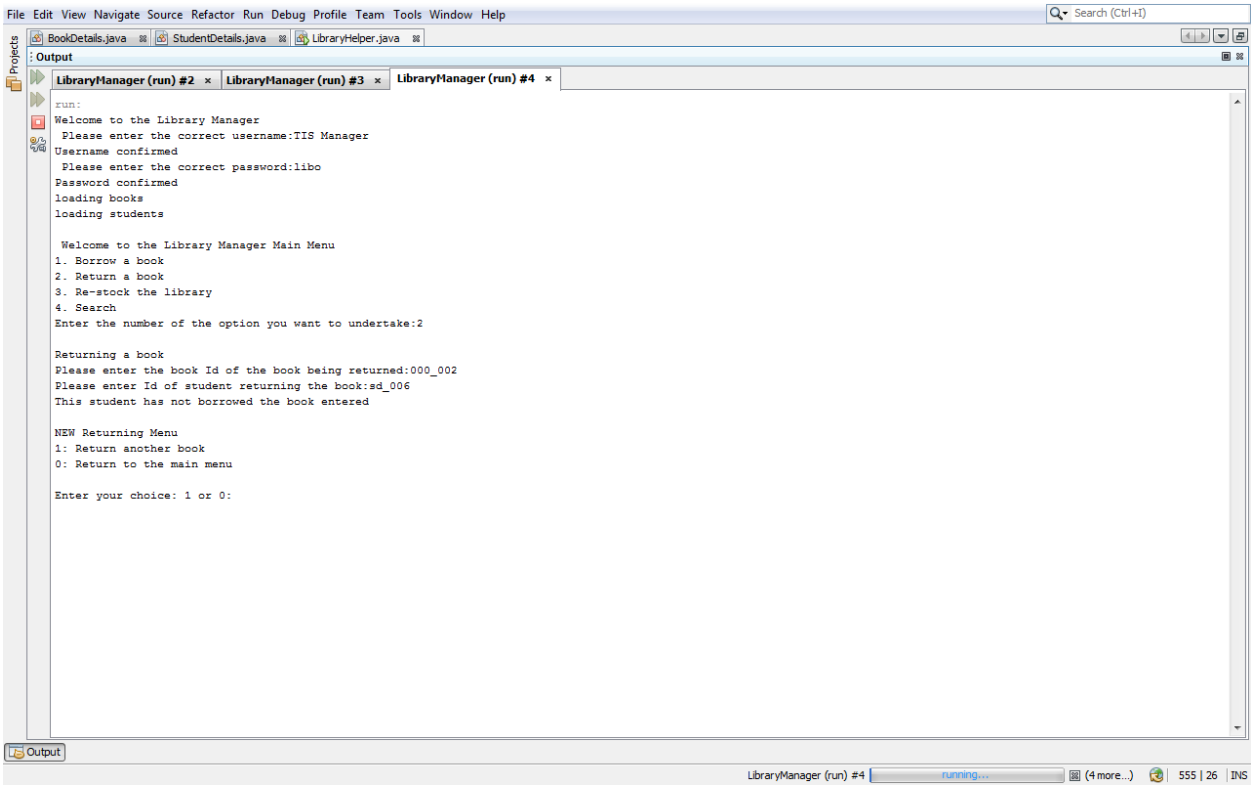
Only when the user has inputted the Id of a book that has been returned will the program allow the user to input the profile of the student returning the book:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
BookDetails.java
StudentDetails.java
LibraryHelper.java
Projects
Output
LibraryManager (run) #2
LibraryManager (run) #3
run:
Welcome to the Library Manager
Please enter the correct username:TIS Manager
Username confirmed
Please enter the correct password:libo
Password confirmed
loading books
loading students
Welcome to the Library Manager Main Menu
1. Borrow a book
2. Return a book
3. Re-stock the library
4. Search
Enter the number of the option you want to undertake:2
Returning a book
Please enter the book Id of the book being returned:-123
Please Book Id must begin with '000_'
NEW Returning Menu
1: Return another book
0: Return to the main menu
Enter your choice: 1 or 0: 1
Please enter the book Id of the book being returned:000_001
Please this book has not been borrowed
NEW Returning Menu
1: Return another book
0: Return to the main menu
Enter your choice: 1 or 0: 1
Please enter the book Id of the book being returned:000_002
Please enter Id of student returning the book:

At this stage, the user must input a student Id that exists, and that borrowed the book being returned. The error messages for these inputs are shown below:

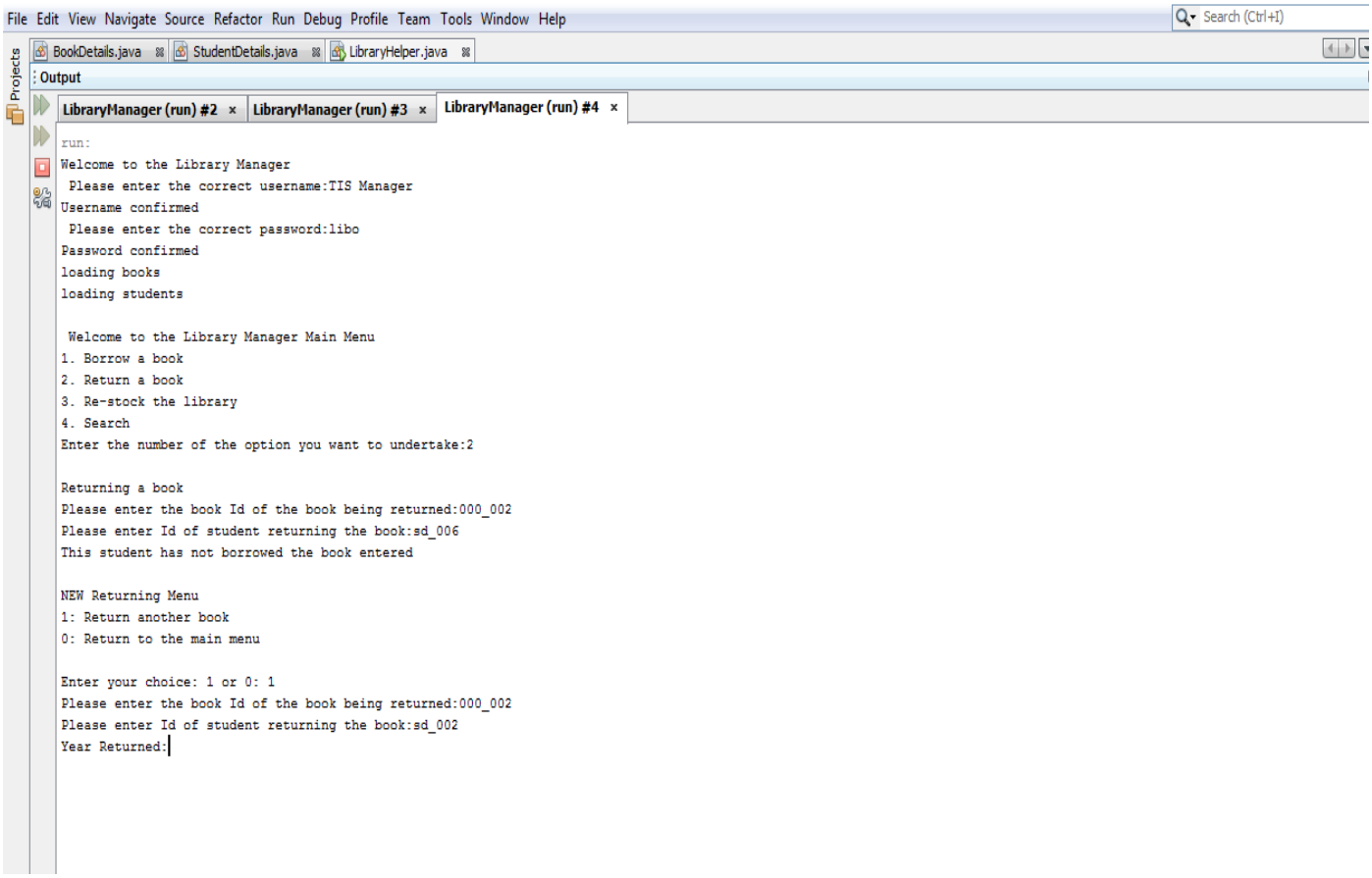


The screenshot above is for when the student Id entered does not exist in the library database.

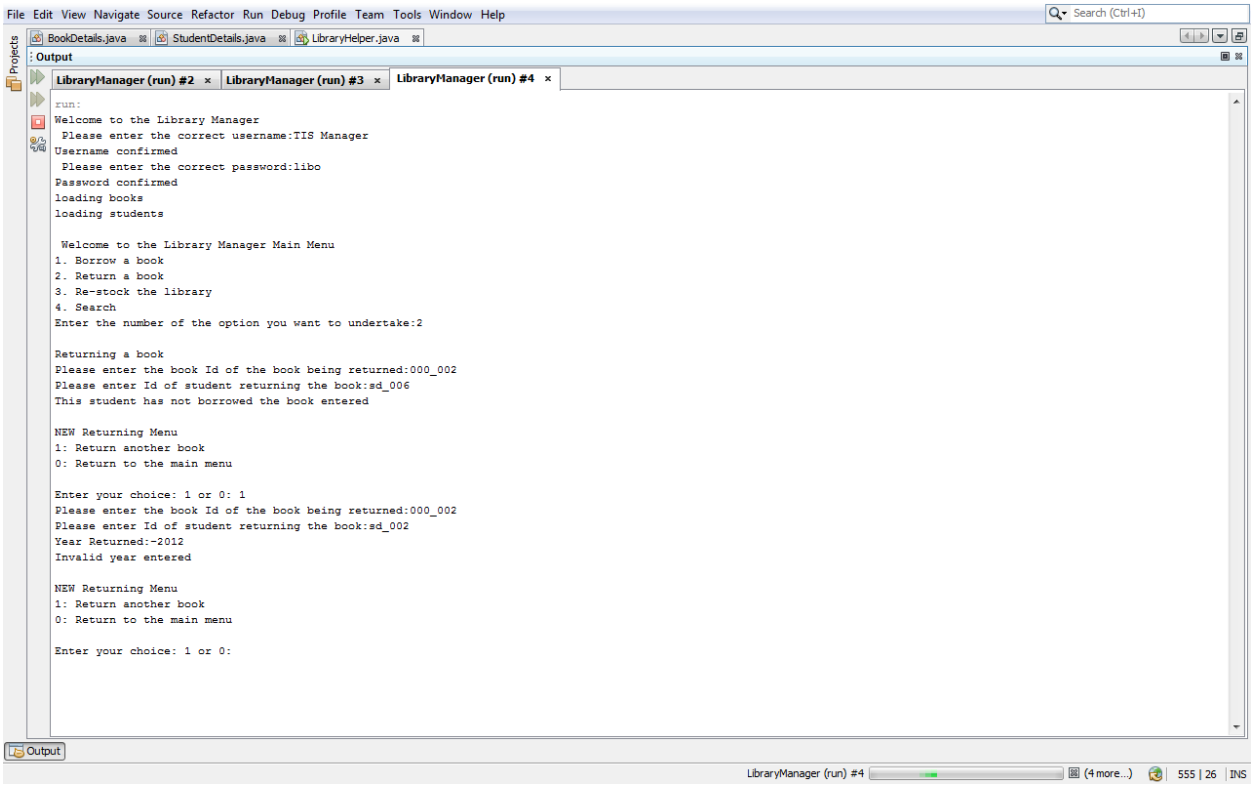


The screenshot above is for when the student has not borrowed the book entered.

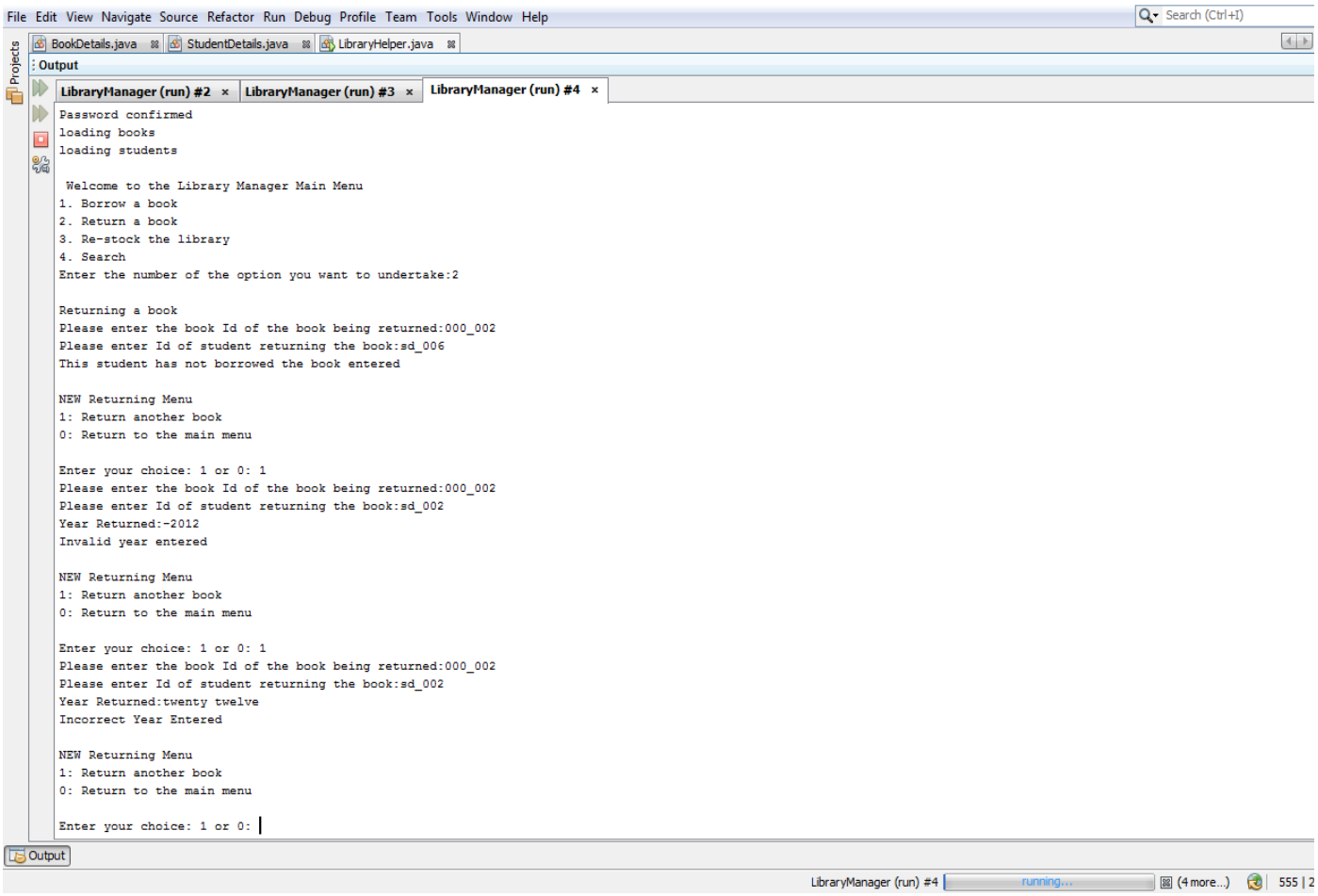
However, when the user enters the student who borrowed the book being returned, the program allows the user to update the profile of the student returning the book:



Also, during this stage, the user must input a year that is from 2012; any other input will be regarded as invalid by the program. The screenshots below show this:

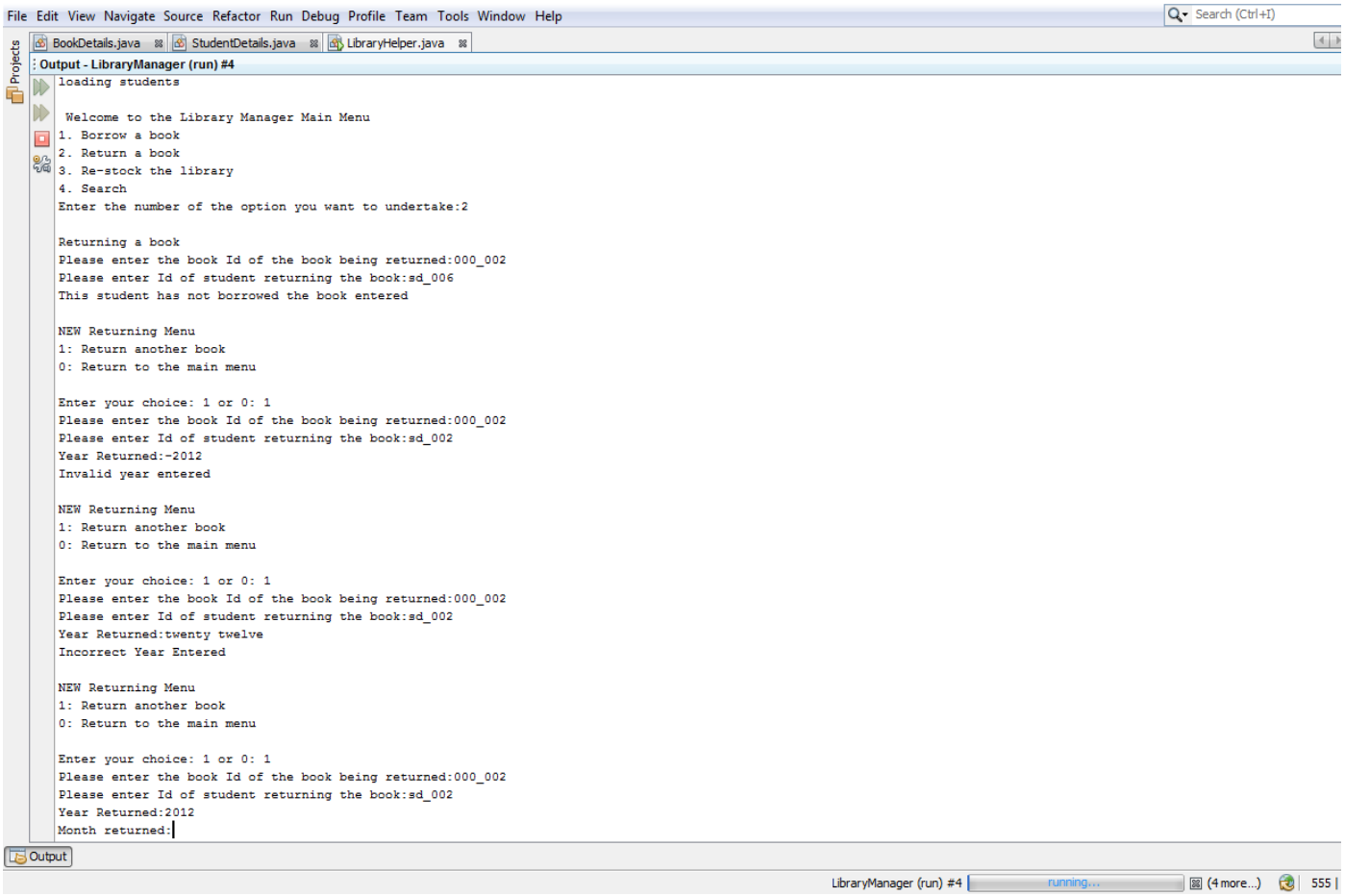


The screenshot above is for when a negative number is inputted.

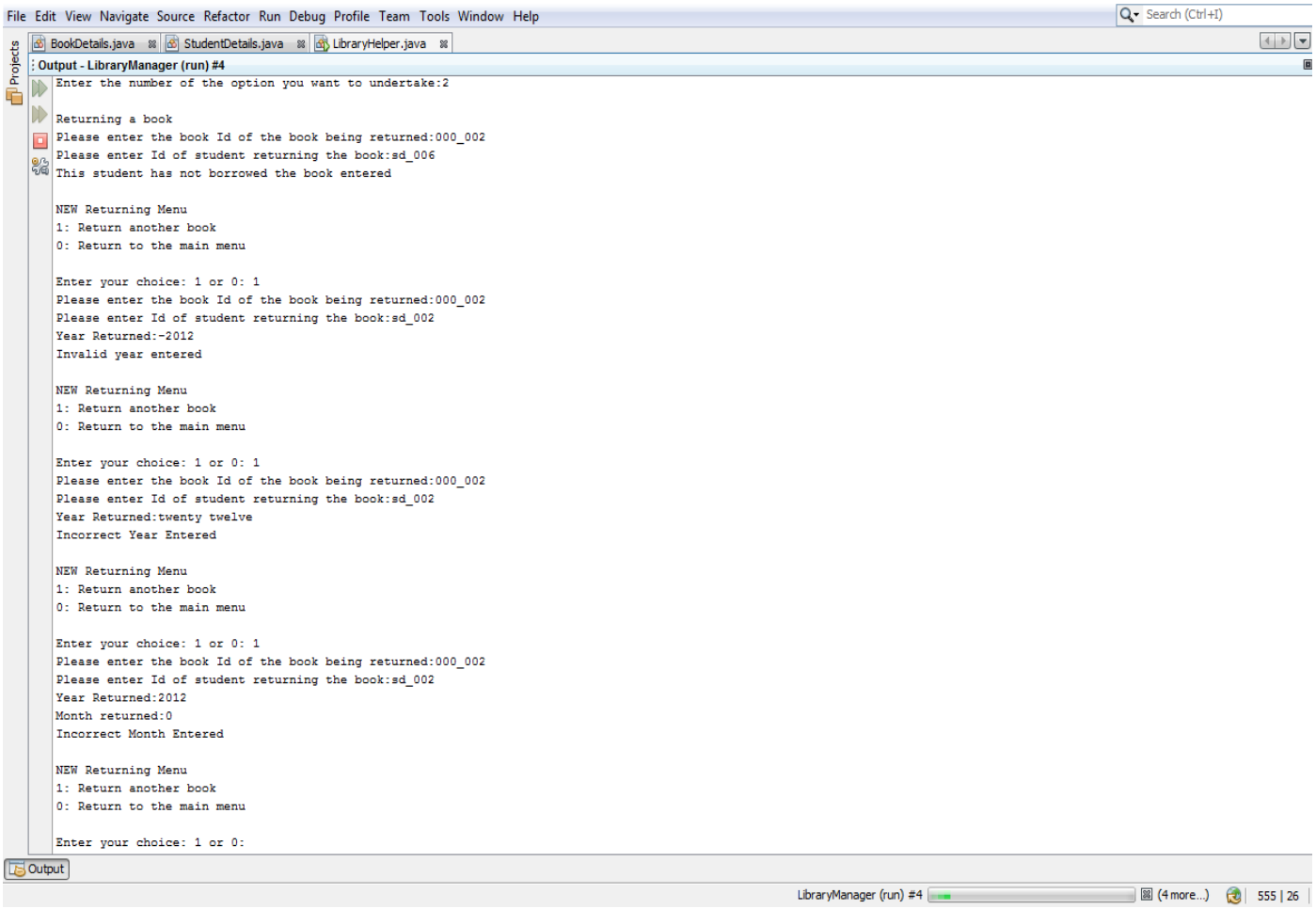


The above is for when letters are inputted in this field.

When the correct input for the year returned is entered, then the program asks the user to input the month the book was returned:



At this point, the correct inputs for this stage are integers ranging from 1-12; any other input apart from this will be considered as erroneous by the program will cause it to display an error message:

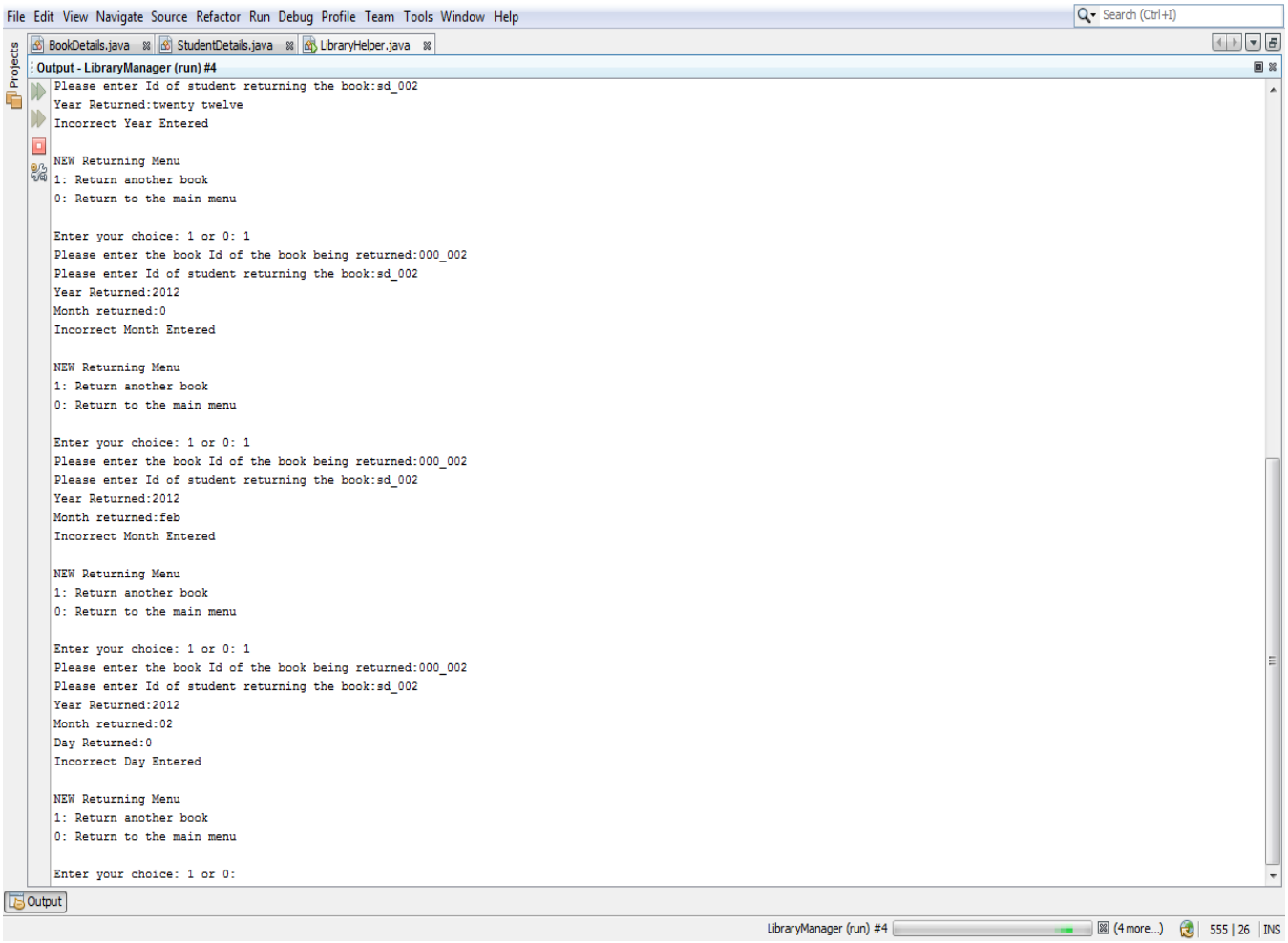


The above is for when zero is inputted for the month returned field.

The above is for when letters are inputted by the user.

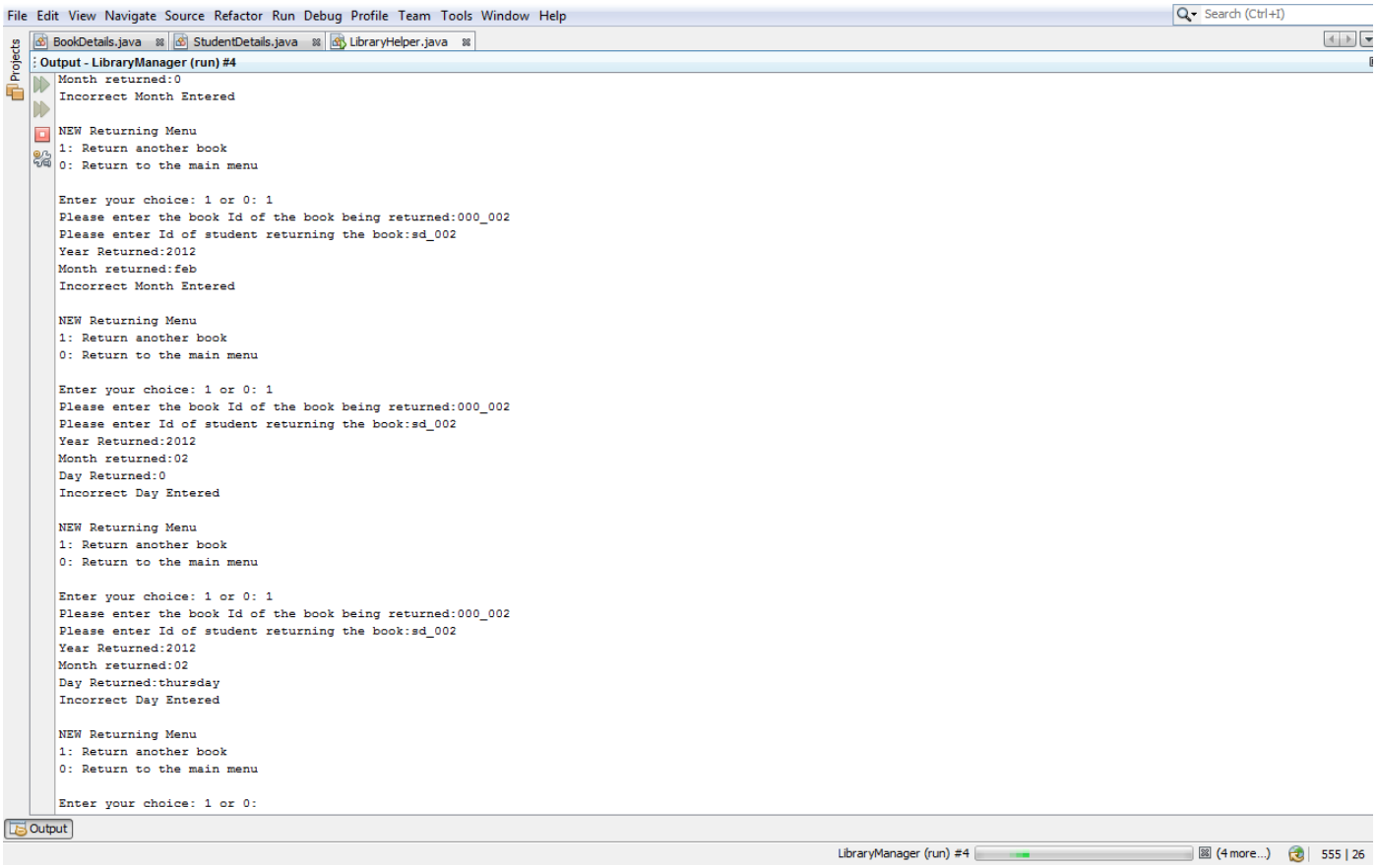
After the input for the month is entered, then the user is now allowed to input the day the book was returned. For this field, the correct inputs are numbers that range from 1 to 31; however, the upper bound for this field depends on the month the book is being returned.

When a number less than zero is inputted for this field, the program screenshot becomes:

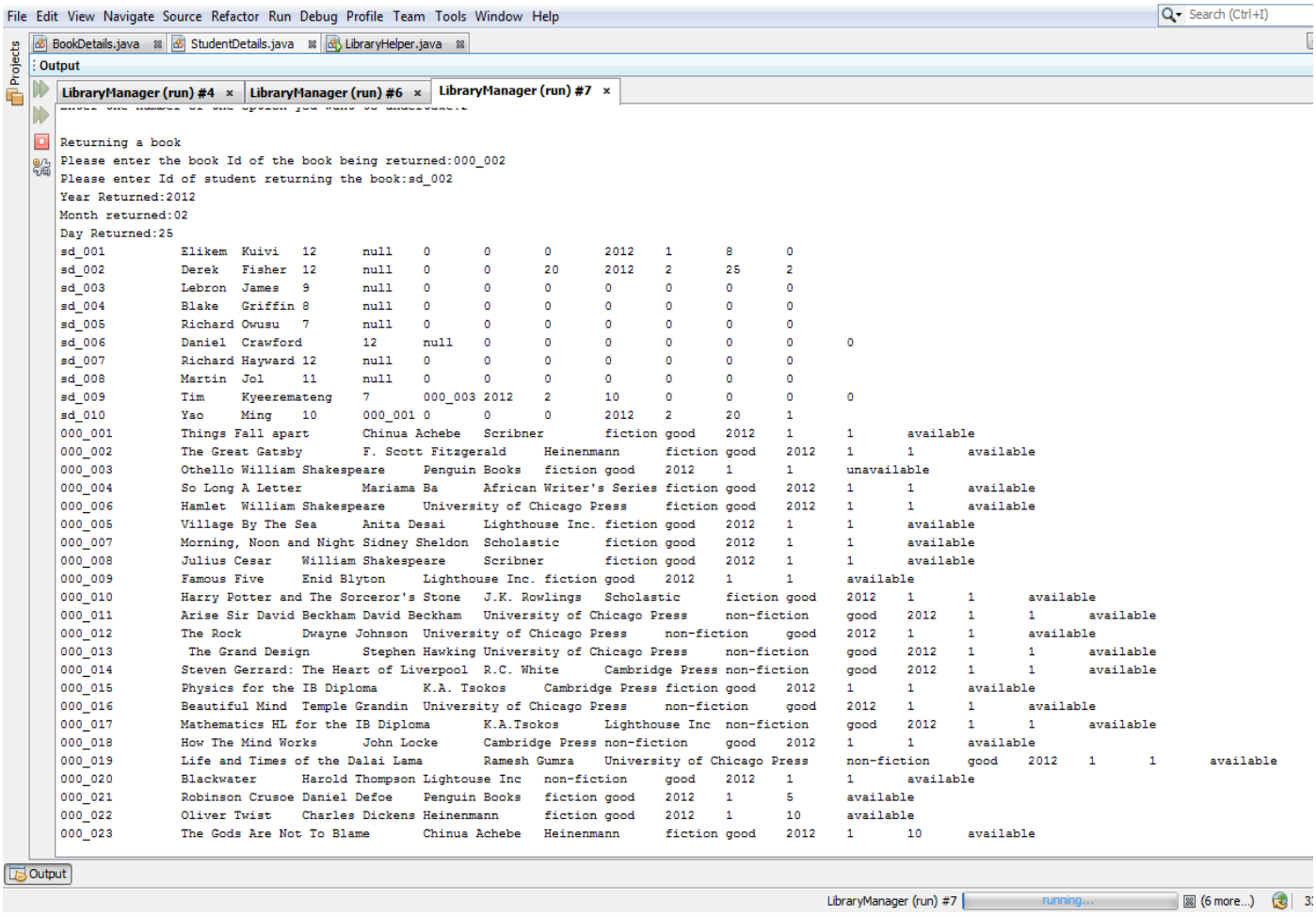


The above is for when zero is inputted in this field.

When letters are inputted in this field, the program screenshot becomes:

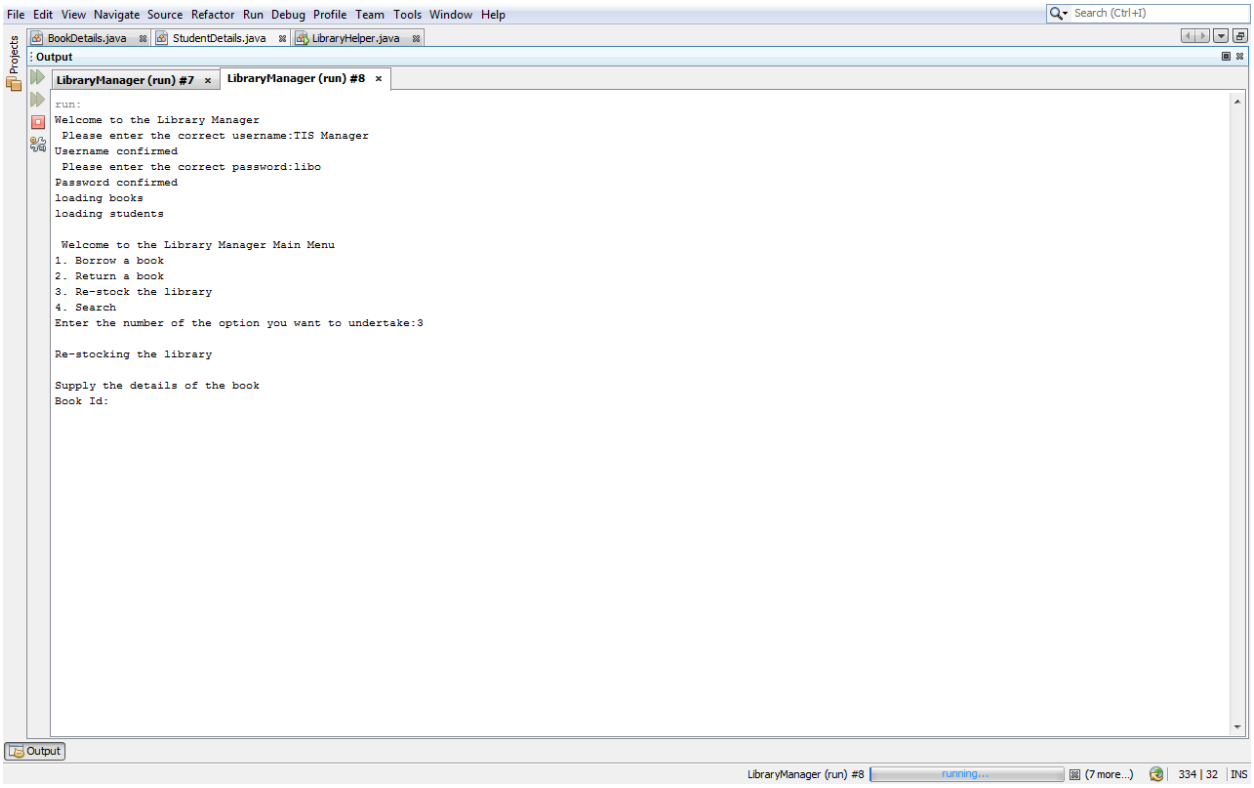


Only when the correct data is inputted for the day the book is returned will the program update the student and book profiles:

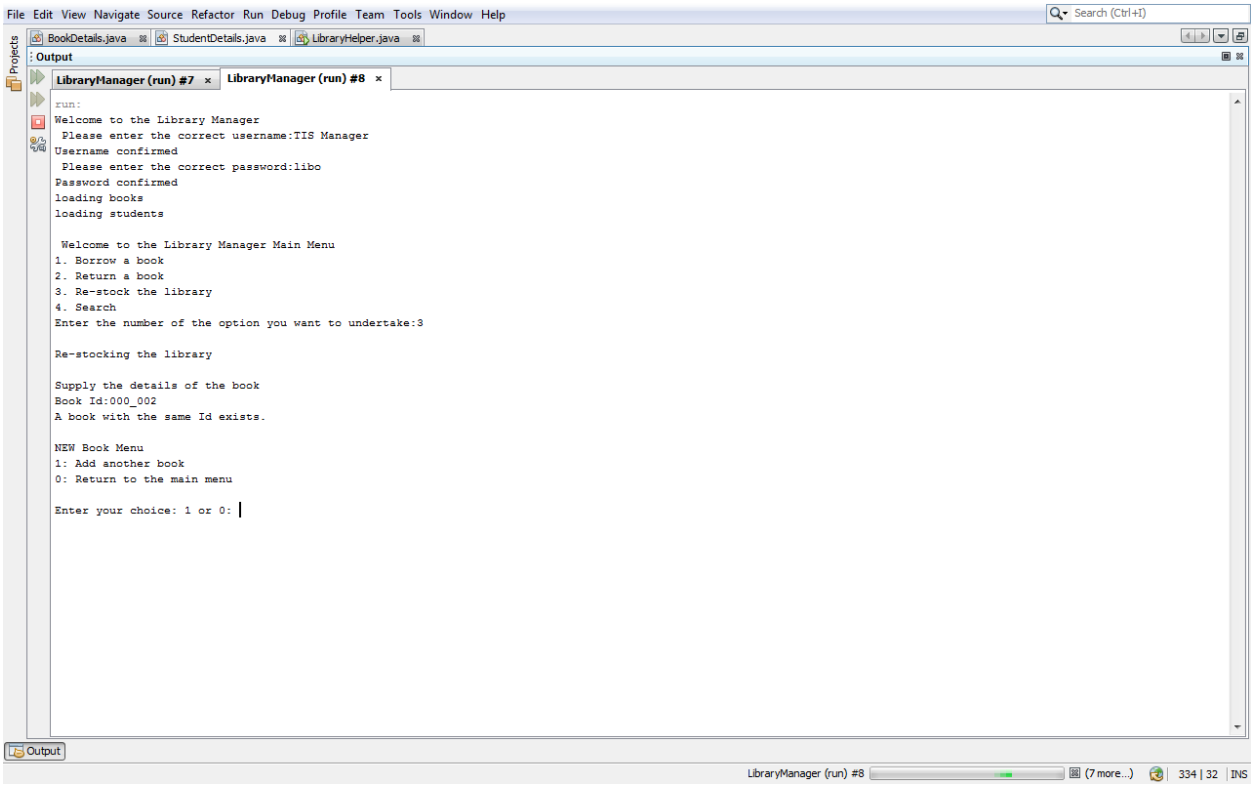


The diagram above shows that the student profile with Id ‘sd_002’ has been updated to show that the student returned the book as the student’s profile now reads ‘null’ for book borrowed. Also the status of the book profile with Id ‘000_002’ has changed to ‘available’ to show that the book has been returned.

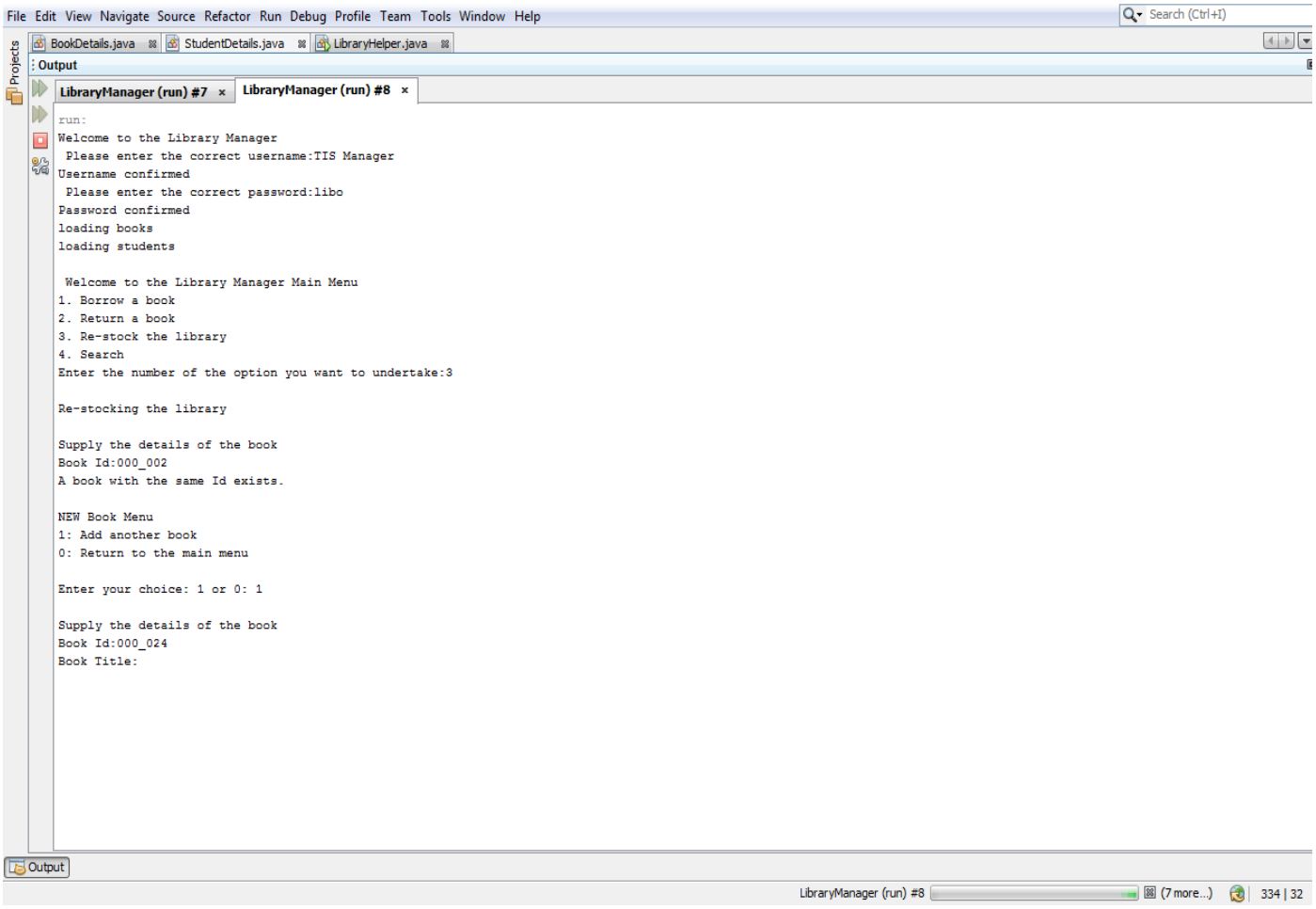
Now when the user enters the re-stocking interface the program looks like:



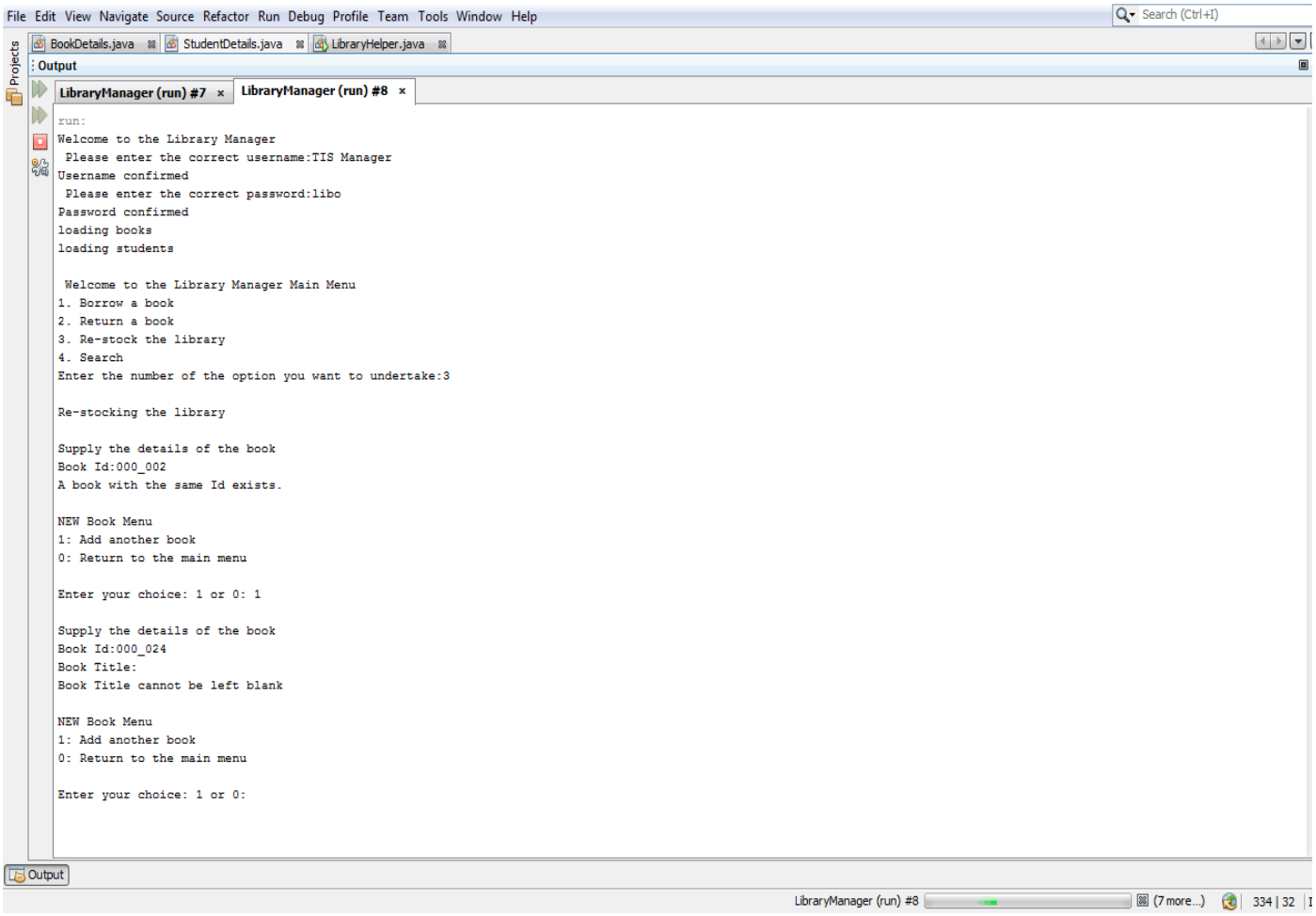
At this stage, the user can only input the a book Id that does not exist, if the user inputs a book Id thats exists, the program shows that:



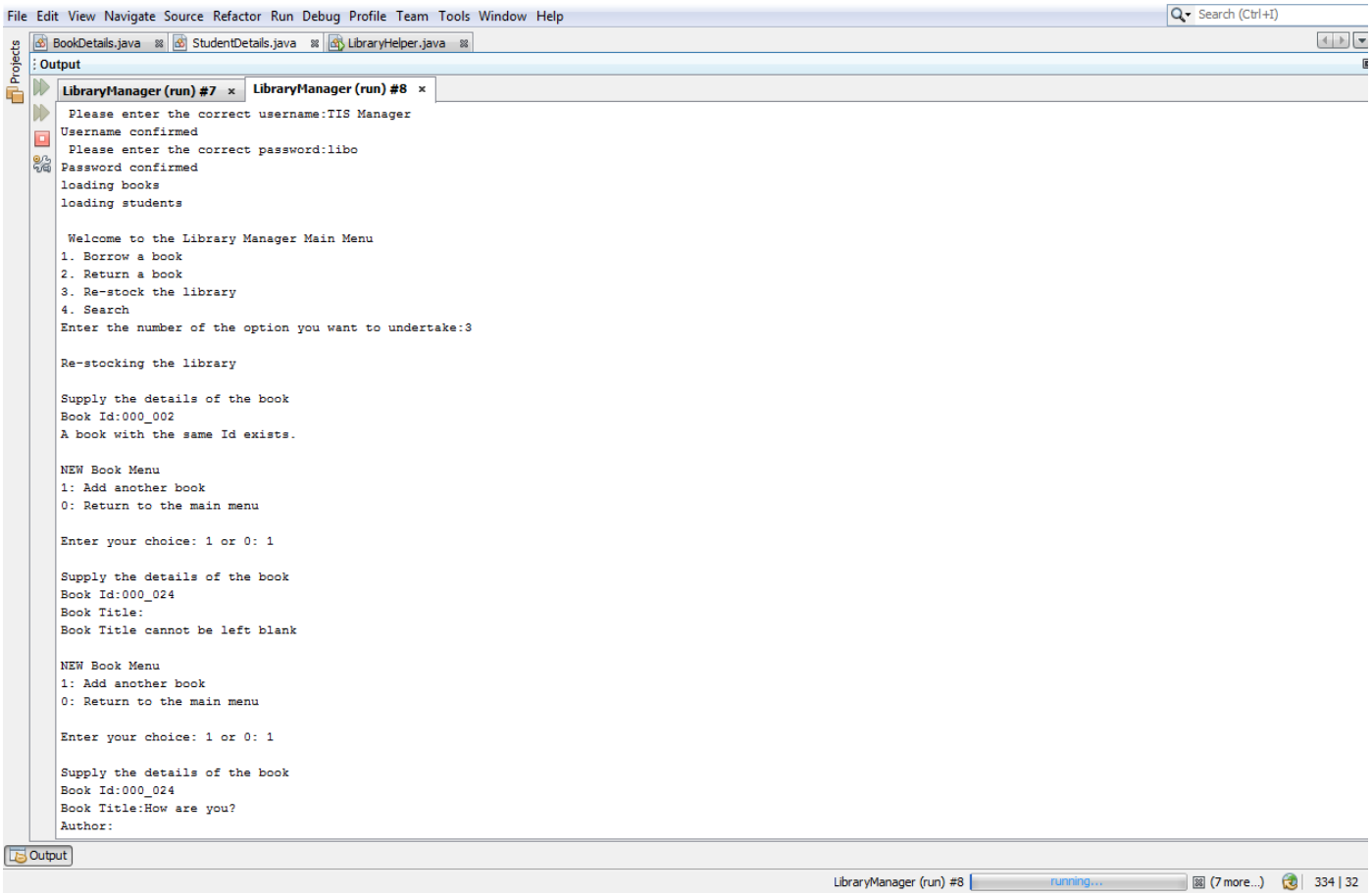
After a book Id that does not exist in the library database is entered, the program then allows the user to input the other details of the book profile being added:



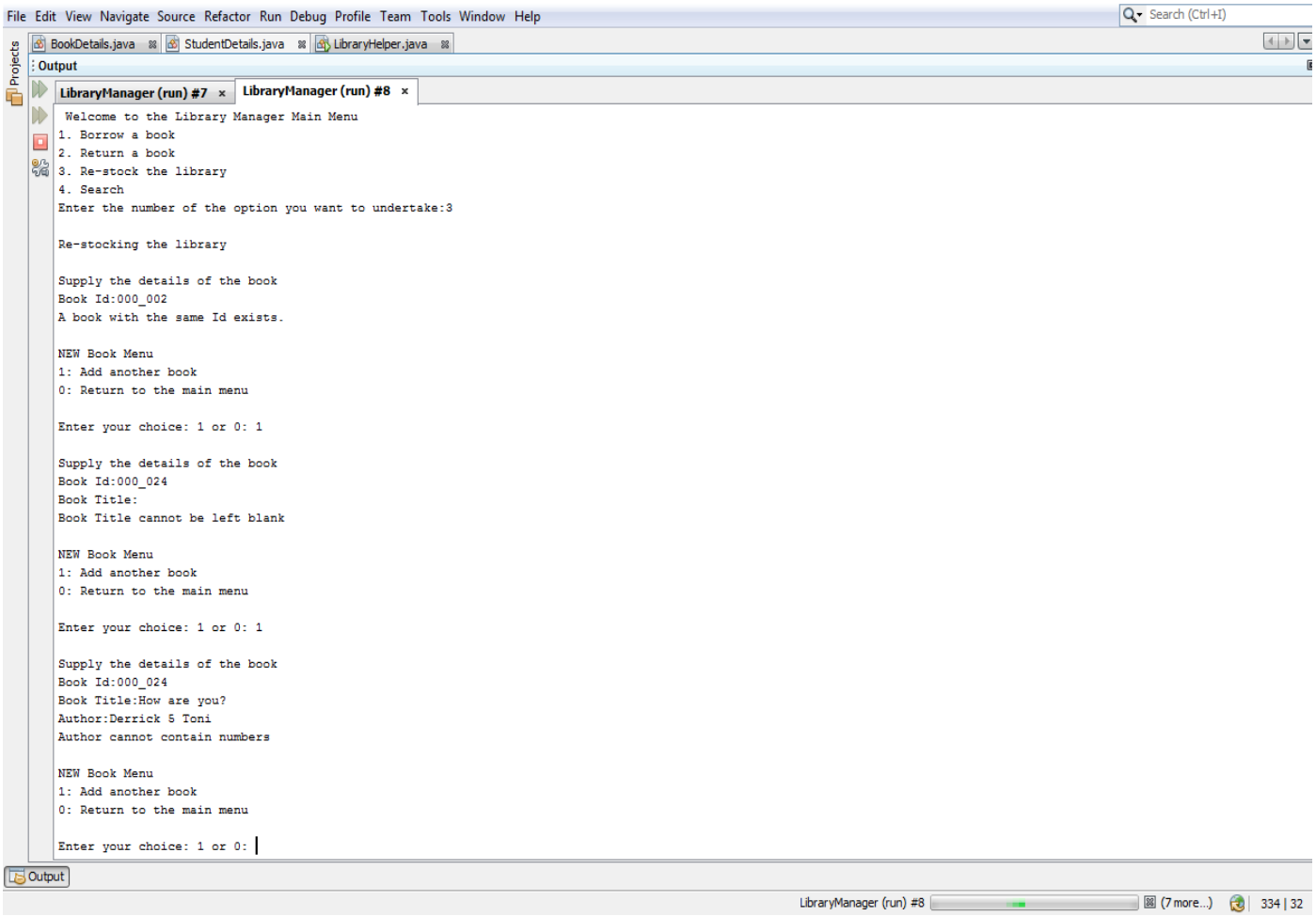
At this stage the program ask the user to input the title of the book being added; at this point, input will only be invalid if the user leaves this space blank:



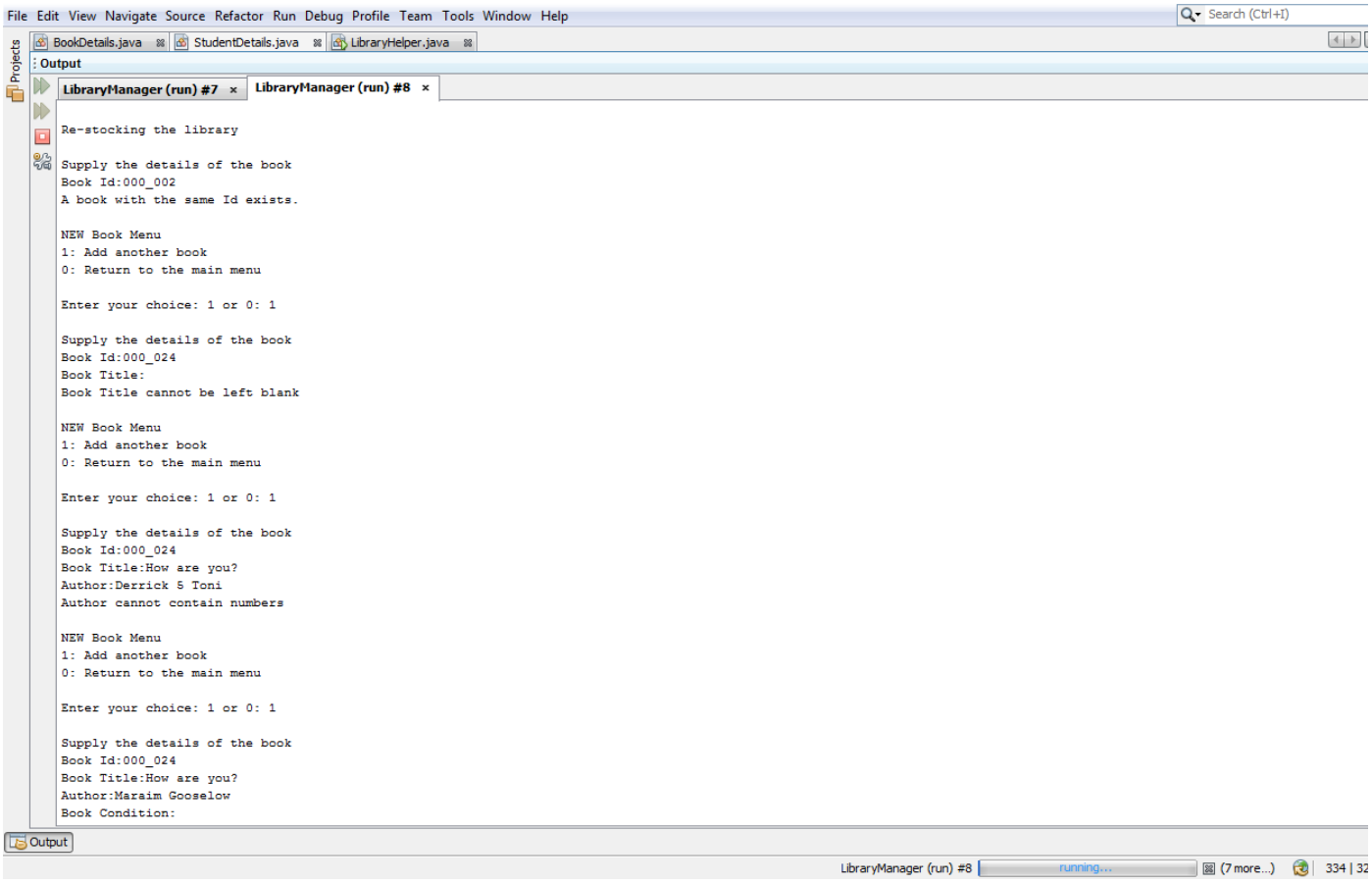
Only when data is inputted for the book Id will the program allow the user to input the author of the book being added:



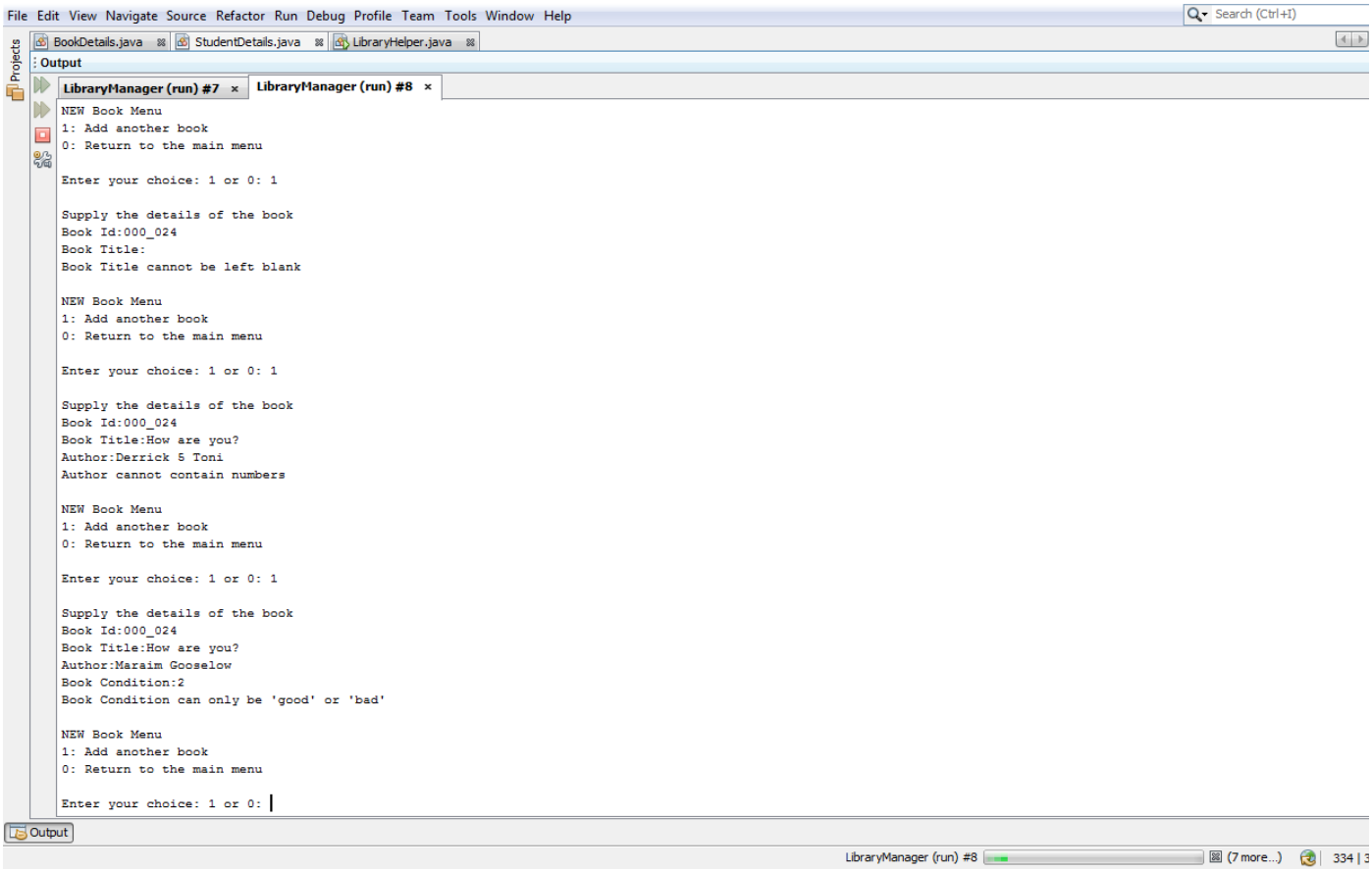
For the author input, the program will only accept string variables that contain no numbers; thus when a number is included in the author's name, the program displays:



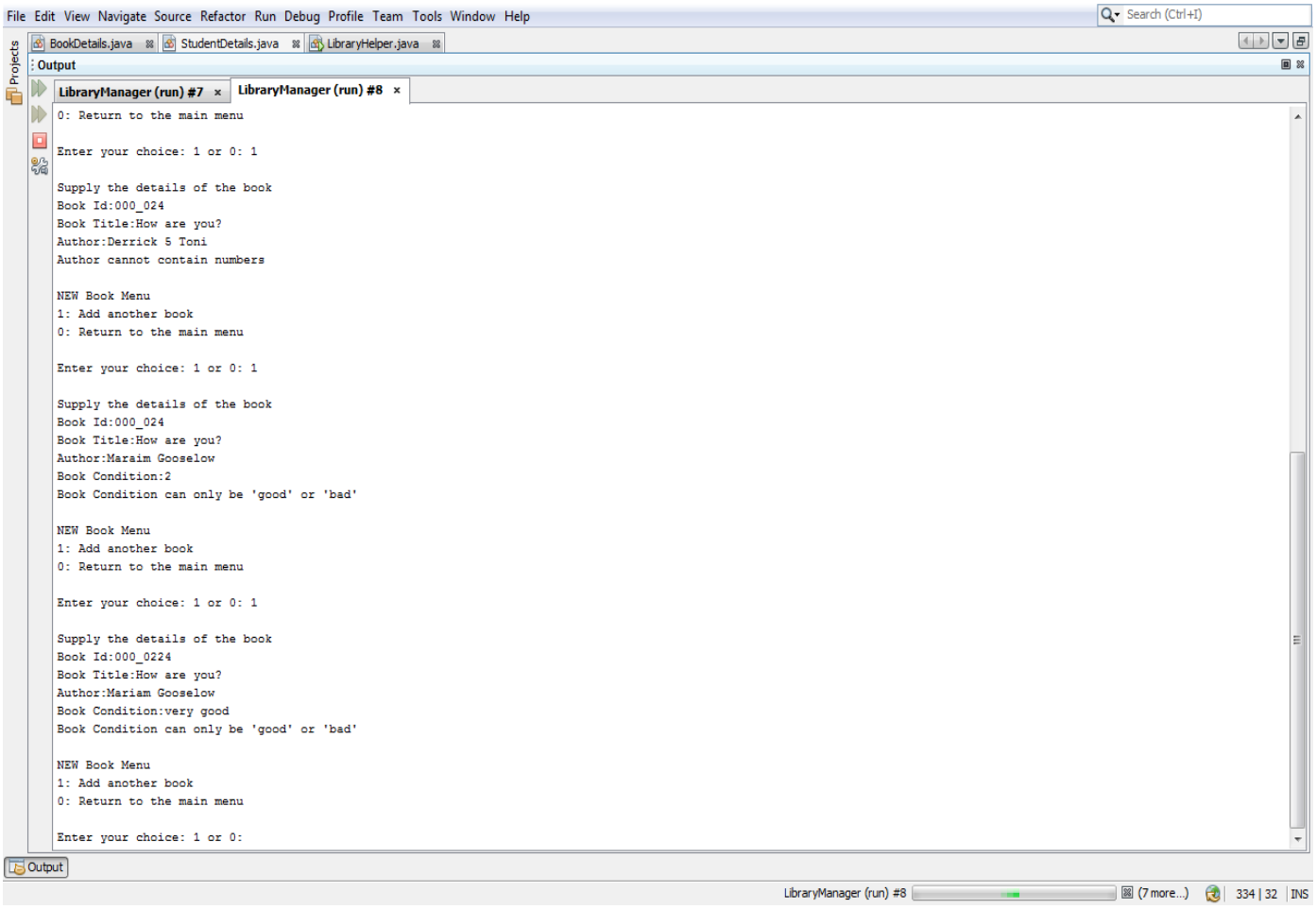
Only when the user inputs the correct data in the author field, without numbers will the program all the user to input the condition of the book:



At this stage, the user can only input ‘good’ or ‘bad’ into this field, any other input will be received as invalid by the program:

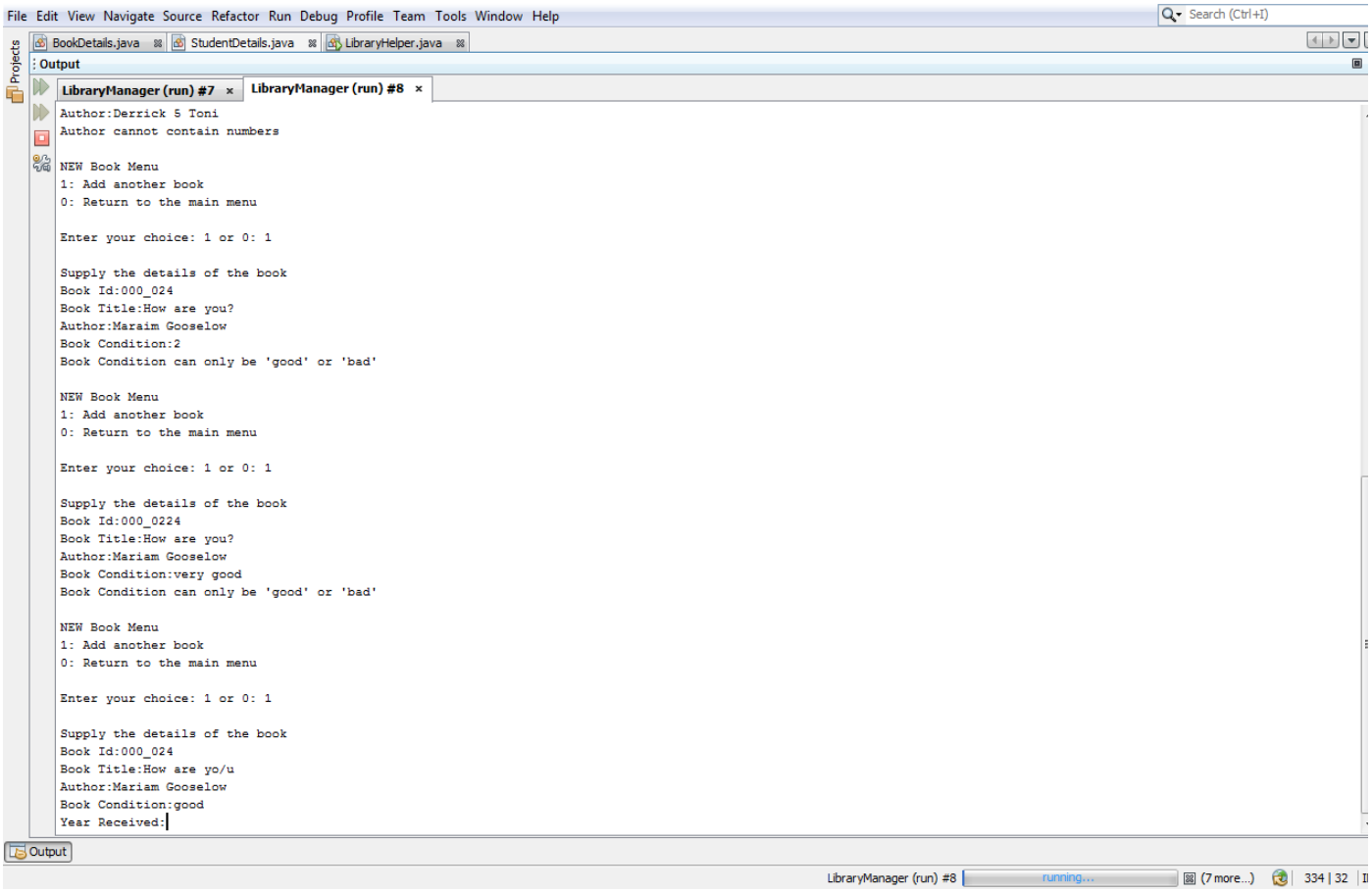


The above is for when a number is inputted in this field.

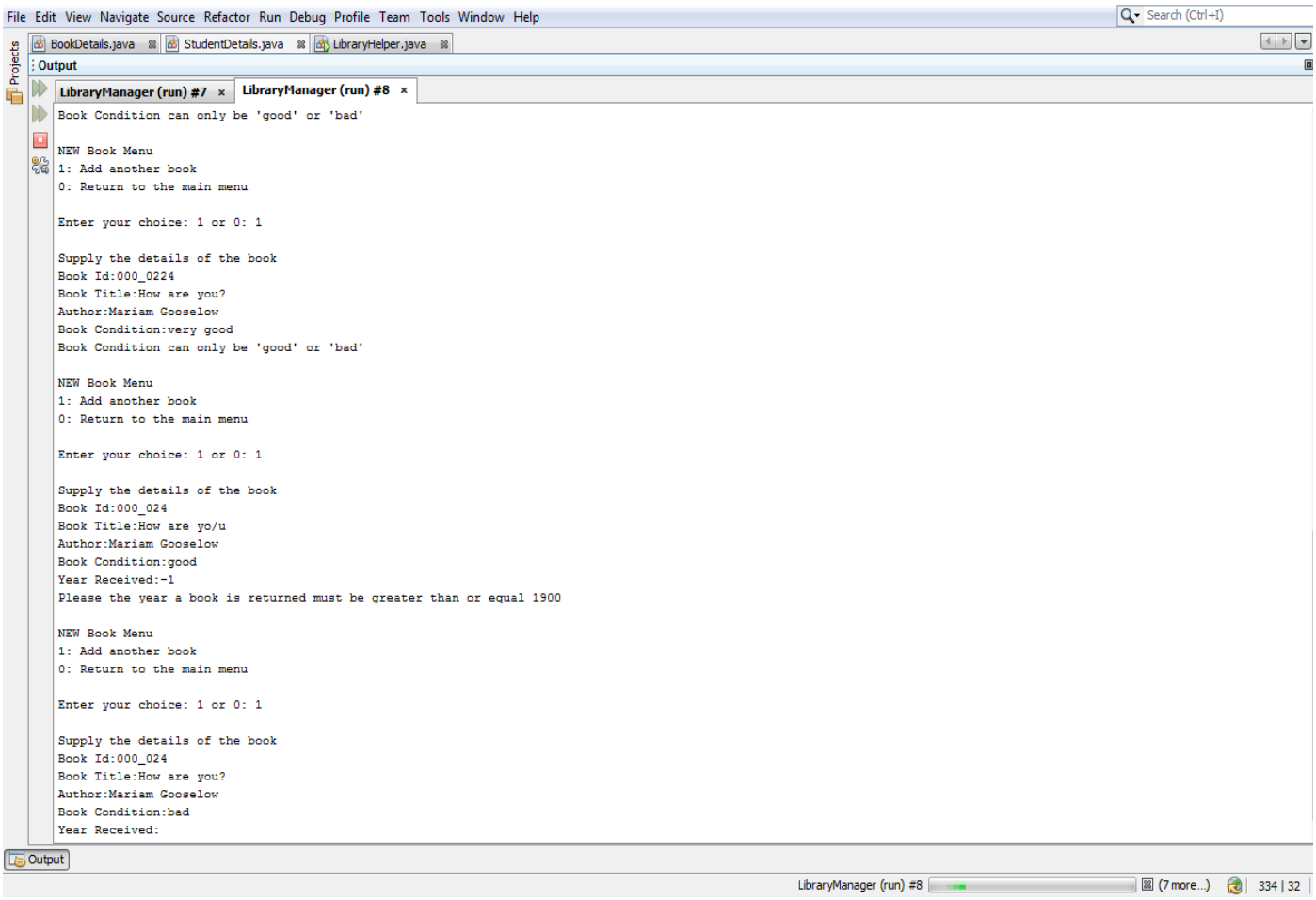


The above is for when incorrect data is inputted by the user.

However, when the correct data s input in this field, then the program asks the user to input the year the added book was received:

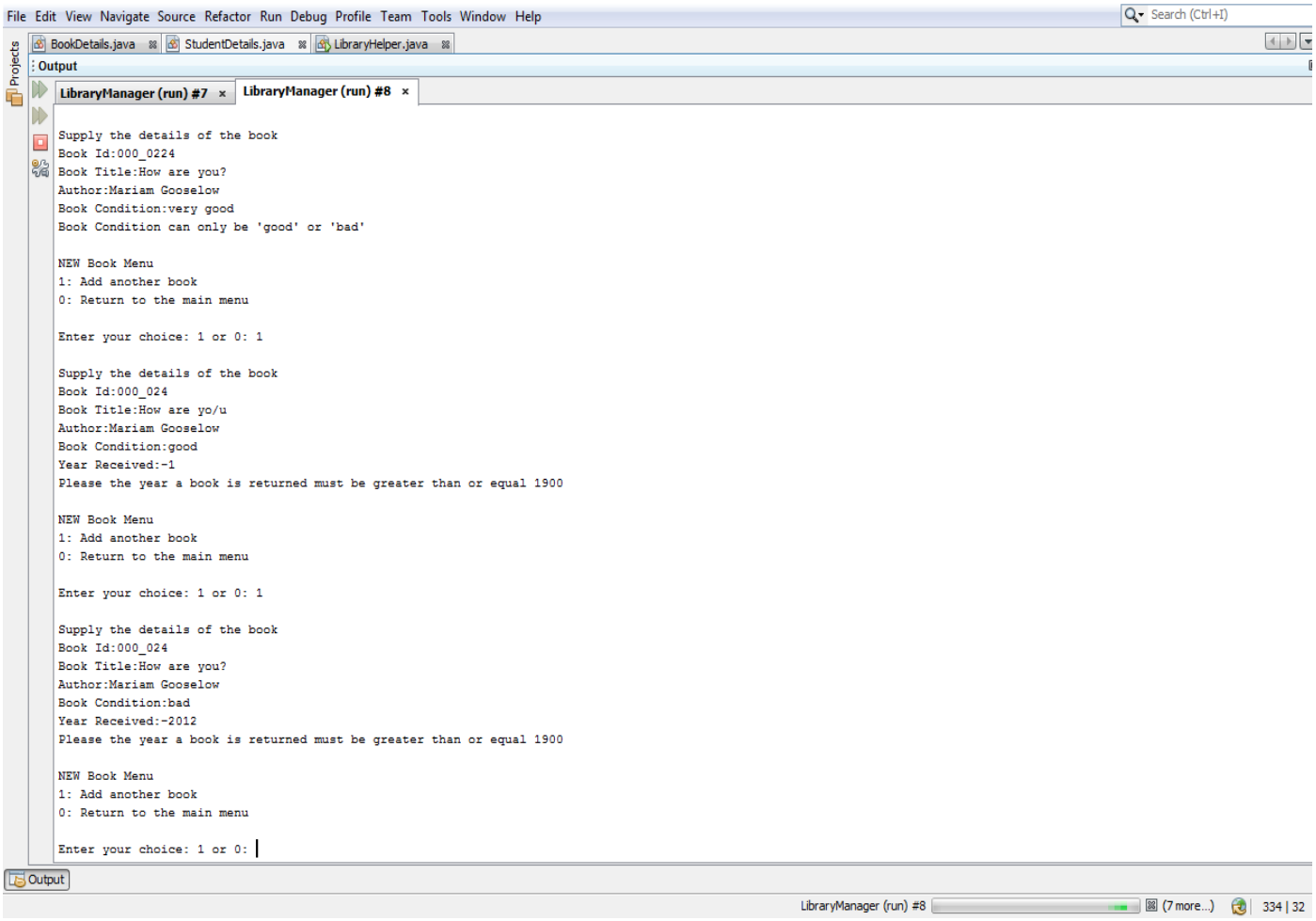


The above is when ‘good’ is entered in the condition field.

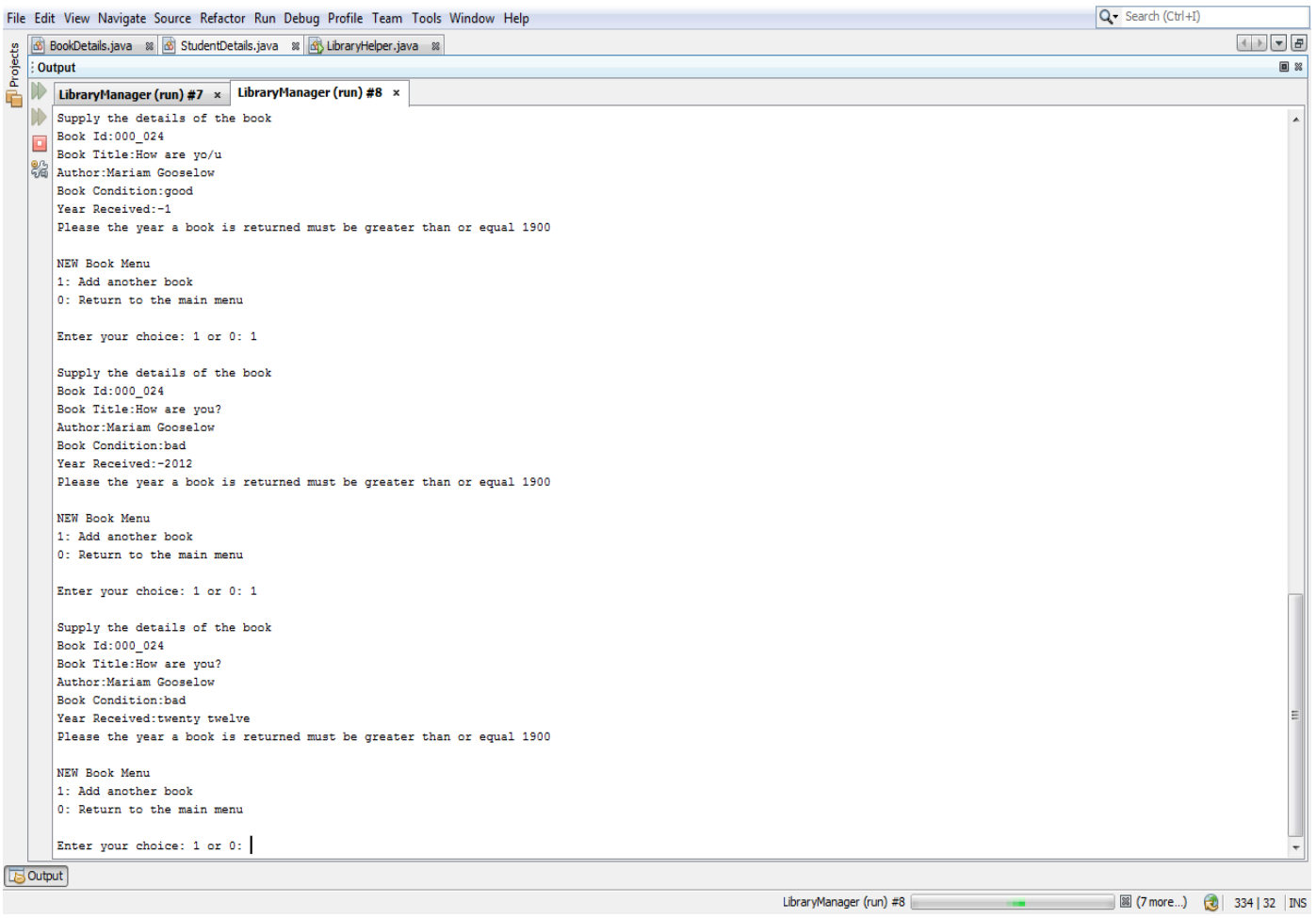


The above is when ‘bad’ is inputted by the user.

Just like the inputs for ‘year due’ and ‘year returned’, the program displays error message when the user inputs negative numbers or letters for this input:



The above is for when a negative number is inputted in this field.



The above is for when letters are inputted in this field.

Conclusion

Program Outline

The solution solves the problem as it allows the librarian to store the details of students and books in the library, and also allows the librarian to edit this data to show that a book has been borrowed or returned. This is done by saving the details of all books in the library, and all students who use the library in separate files, and updating the corresponding data in the fields for each book and student record to show that a borrowed book is returned, a new book has been added to the library, or a student has borrowed a certain library book.

From the librarian and students' point of view the program is faster to use than the old system, and produces more accurate calculations than the old system. Also the librarian finds that the program is easier to use than the old system, and is less tedious to use than the old system. The programming for this solution is modular, and also makes use of user-defined methods.

Although the code for this program was relatively long, the most of the code inputted was uncomplicated. The program did not make use of sorting, however, the program contains may instances of searching.

Effectiveness and Efficiency

The program achieves all the objectives set for it earlier during the criteria for success. However, since the program searches through every single record sequentially when Id's of students and books are inputted and searched for in the library database, the program takes a long time to finish searching, when the files for student and book details contain a lot of profiles. Especially, when the profile to be searched for is the last record in the database, the search speed will be equal to all the profiles in the library database.

Improvements

Since the program searches for data in the library database sequentially, and since this can make the program slow when it contains large amounts of data, the program can be improved to contain sorting methods. This will make the program sort all the data in the library database according to their Id's, so that when it contains large amounts of data, a binary search can be used to improve the overall speed of the search.

Also, the program can be improved by adding a graphical interface to the program so that it will be more user-friendly.

Alternative Approaches

The design methodology applied in creating the solution was adequate enough to create the desired solution to the program. As a modular design was adopted in the creation of the program, it made it easy to debug the program during the testing stage. Also, the sending of the prototype to the librarian made me more aware of the requirements of the librarian, and thus caused me to create a program that was tailored to the librarian's needs.

However, the design methodology adopted was not all that adequate since a better strategy could have been adopted to make the solution more suitable to needs of all those who will interact with the program, and this includes students and the accounts office.

Firstly, the prototype could have also been sent to the students and the accountants, so that I will also be aware of the things they require from the program; this would have allowed me to create a program that me needs of the librarian, students, and accountants, instead of creating a one-sided program that only concentrates on meeting the needs of the librarian.

Also for the design of the program, the program should have been made to give students, accountants, and the librarian separate accounts; this would have been better, since information like the one produced when lists are outputted by program, will no more be done by the librarian, but will be done by the students and accountants instead, as they are the ones who need that kind of information.