SCHOOL OF COMPUTING

FACULTY OF ENGINEERING

UNIVERSITY TECHNOLOGY MALAYSIA

OBJECT ORIENTED PROGRAMMING

(SCSJ2154)

SEMESTER II 2018/2019

MINI PROJECT DOCUMENTATION

TITLE: LIBRARY MANAGEMENT SYSTEM

By

NG JIA QI	(A17CS0132) - GROUP LEADER
JOVITA MAULINA YUSUF	(A17CS0255)
TAN SEE JOU	(A17CS0218)

SECTION 09

LECTURER:

MDM LIZAWATI BINTI MI YUSUF

Table of Contents

A. Task Distribution

The project is separated into three modules which represent the main components of a library, which are books, users and study rooms respectively. Since our system is target for librarian and students, thus, the user perspective is separated into two parts. After the discussion, we had separated our tasks evenly.

Member	Tasks
Ng Jia Qi	GUI User - staffs class Exception Handling Study Room User - students class
Jovita Maulina Yusuf	User - students class Study Room
Tan See Jou	Books Main class

A. Project Description

The requirements of Object-Oriented Programming(SCSJ2154) Project is to construct a system that implements Java technology and object-oriented programming concepts. Our system for this project is Library System. The main objective of our system is to provide a convenient and efficient way for librarians to manage and students to enjoy the services. Therefore, our target users are both librarians and students, and both of them have different functions to access.

The services provided in our system including book borrow service and study room service. For book borrowed service, there are only three types of books provided, which are novel, reference book and journal. For each type, there are 3 books, and the quantity is only one for every book. For study room service, there are only two types, which are the individual room and group discussion room. For both librarians and students, they have different functions to access the services. In the system, there are simple databases that store the details for users, for example, name, user ID, password, contact number and so on. Thus, before the user can access any functions, they need to login by input their user ID and password as well. Once inserted, the system will trace their ID and direct them to a different interface.

From **librarian's perspective**, the functions provided in our system included:

- ❖ display booklist
- ❖ search book
- ❖ add new book
- ❖ remove book
- ❖ display student list

Librarians can choose any of the function to access.

For function **display booklist**, the librarian can check the current books in the library which enable them to decide what type of book need to add from time to time.

For the function **search book**, the librarian can search the current book available by inserting the book titles. All the details of that book will be displayed once the book is found.

For function **add new book**, the librarian can add details for a new book by inserting all related information of the book, for example, book title, ISBN (International Standard Book Title), author, type and so on. Different type of book has different information to insert.

For function **remove book**, the librarian can remove the book that is no longer available from the list by inserting the ISBN of the book. With this function, the librarian can update the booklist from time to time.

For function **display student list**, the librarian check the list of the students with detailed information and the list of books they have borrowed.

From the **student's perspective**, the functions provided in our system included:

- ❖ search book
- ❖ borrow book
- ❖ return book
- ❖ check borrowed booklist
- ❖ booking study room
- ❖ check booking status
- ❖ display study room list

Students can choose any of the function to access.

For the function **search book**, students can search the book by inserting the book title. Once the book is found, all the details of the book will be displayed.

For function **borrow book**, students can borrow the book by inserting the ISBN of the book. If the book not found or borrowed by others, the system will display book not found. The limit of the book borrowed for each student is 5, and once the student had borrowed 5 books, he/she is not allowed to borrow any more, until he/she returns.

For function **return book**, students can return the book by inserting the ISBN as well. Once the system found that the ISBN inserted is matched with one of the books in the borrowed list of the students, it will remove that book from the list. If not found, an information message will be displayed.

For function **check borrowed booklist**, students can check the books that they had borrowed and have not returned yet. This function enables the students to check if they had forgotten to return the book and also allowed them to check the book that they had successfully borrowed.

For function **booking study room**, students can book a room either individual room or group room for study. In individual room, only one student allowed to use the room, and for the group room consist of maximum of 4 people in the room.

For function **check booking status**, students can check the room booked in the session. If there is no room booked, the system will also tell users about that.

For function **display study room list**, students can browse through the list of study room provided in the library. The information displayed includes room number, room type, and the status which shows whether room is available.

B. Class Diagram

- Overall UML Diagram

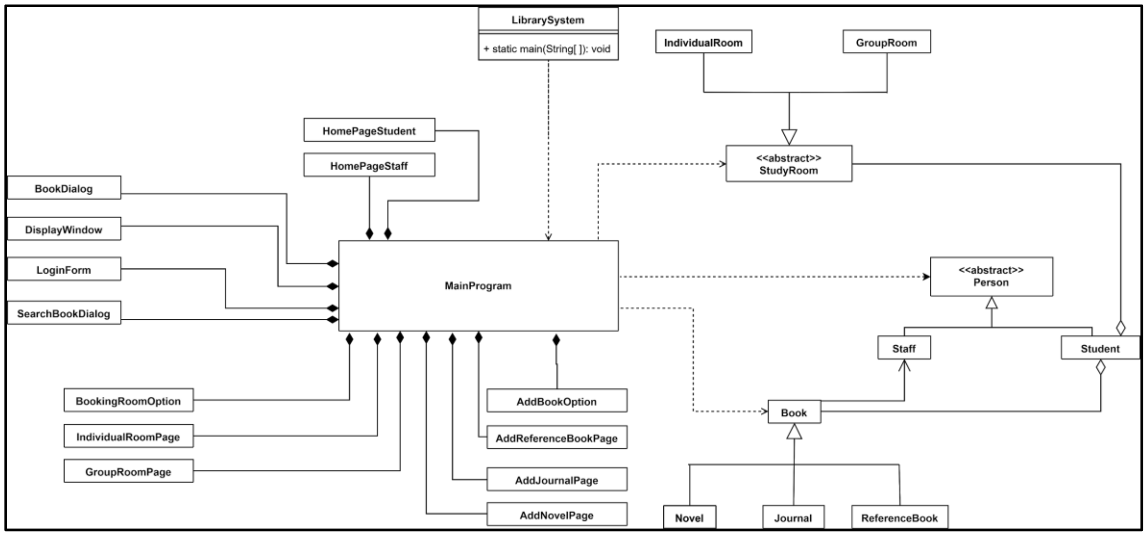


Diagram 1 show the overall UML for the Library System.

- StudyRoom Class

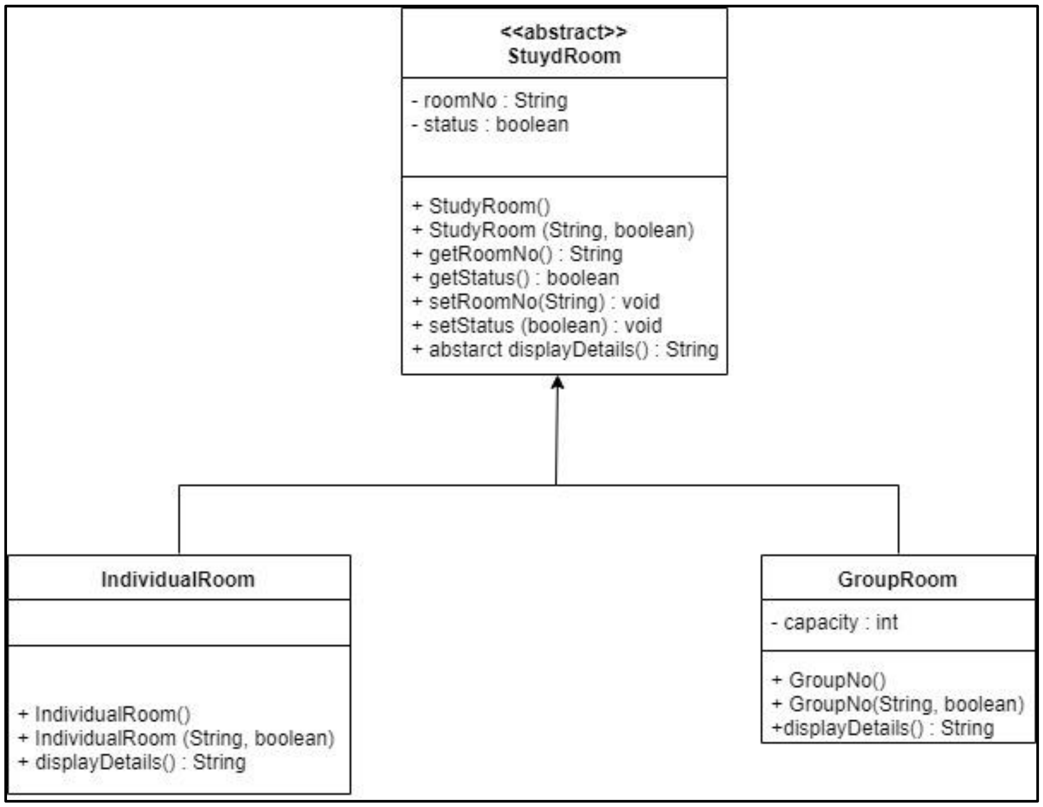

<<abstract>>
StuydRoom
- roomNo : String
- status : boolean
+ StudyRoom()
+ StudyRoom (String, boolean)
+ getRoomNo() : String
+ getStatus() : boolean
+ setRoomNo(String) : void
+ setStatus (boolean) : void
+ abstarct displayDetails() : String
IndividualRoom
+ IndividualRoom()
+ IndividualRoom (String, boolean)
+ displayDetails() : String
GroupRoom
- capacity : int
+ GroupNo()
+ GroupNo(String, boolean)
+displayDetails() : String

Diagram 2 show the StudyRoom UML.

❖ Book class

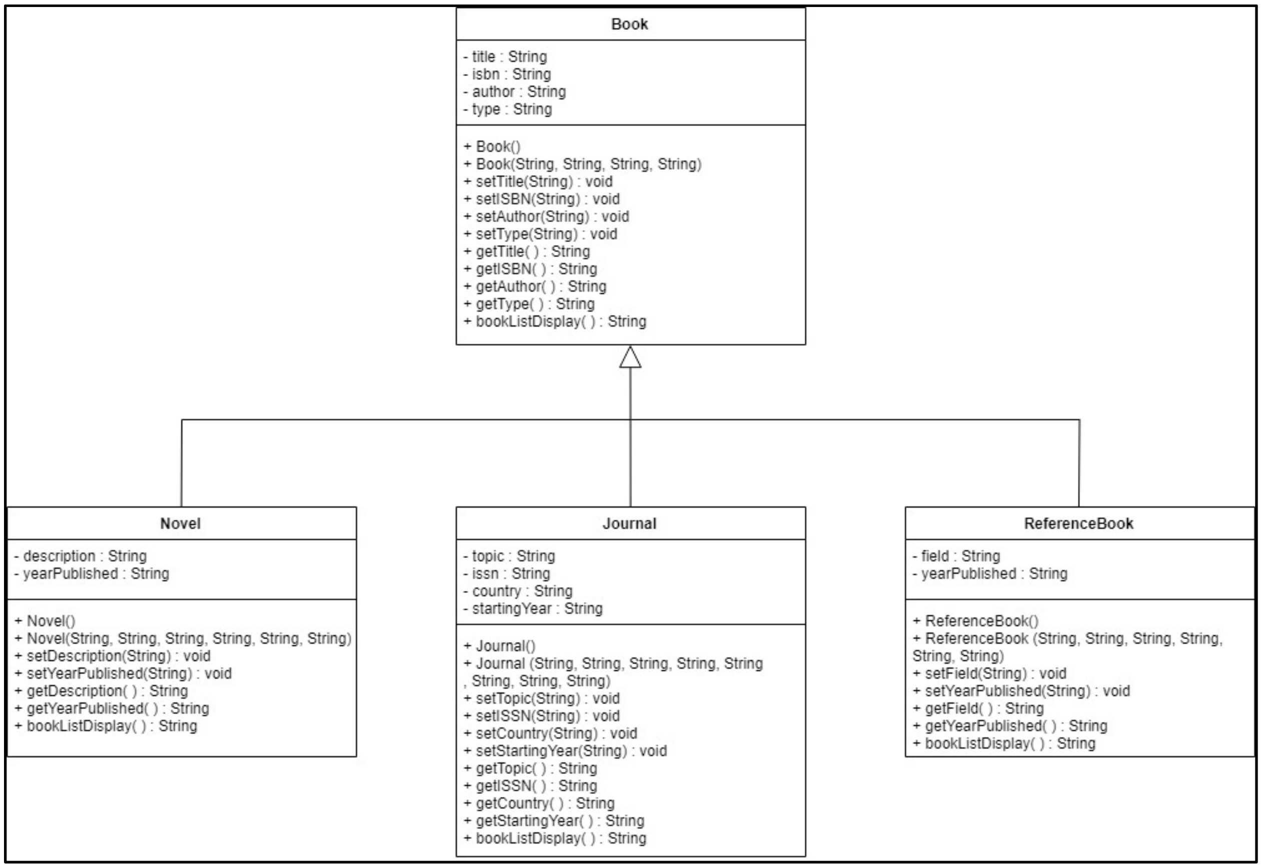
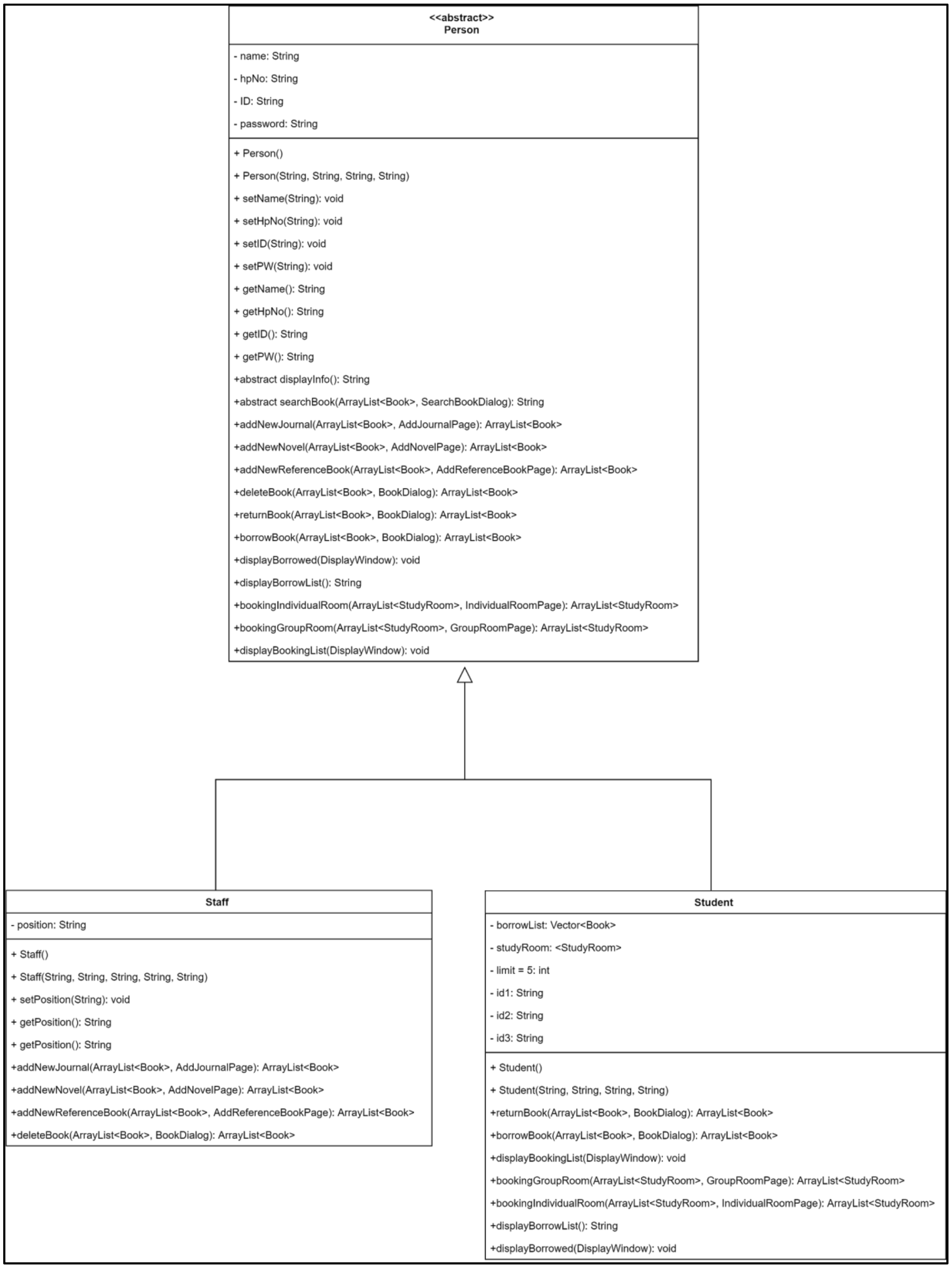

Book
- title : String
- isbn : String
- author : String
- type : String
+ Book()
+ Book(String, String, String, String)
+ setTitle(String) : void
+ setISBN(String) : void
+ setAuthor(String) : void
+ setType(String) : void
+ getTitle( ) : String
+ getISBN( ) : String
+ getAuthor( ) : String
+ getType( ) : String
+ bookListDisplay( ) : String
Novel
- description : String
- yearPublished : String
+ Novel()
+ Novel(String, String, String, String, String, String)
+ setDescription(String) : void
+ setYearPublished(String) : void
+ getDescription( ) : String
+ getYearPublished( ) : String
+ bookListDisplay( ) : String
Journal
- topic : String
- issn : String
- country : String
- startingYear : String
+ Journal()
+ Journal (String, String, String, String, String
, String, String, String)
+ setTopic(String) : void
+ setISSN(String) : void
+ setCountry(String) : void
+ setStartingYear(String) : void
+ getTopic( ) : String
+ getISSN( ) : String
+ getCountry( ) : String
+ getStartingYear( ) : String
+ bookListDisplay( ) : String
ReferenceBook
- field : String
- yearPublished : String
+ ReferenceBook()
+ ReferenceBook (String, String, String, String,
String, String)
+ setField(String) : void
+ setYearPublished(String) : void
+ getField( ) : String
+ getYearPublished( ) : String
+ bookListDisplay( ) : String

Diagram 3 show the Book class UML.

❖ Person Class



<<abstract>>
Person
- name: String
- hpNo: String
- ID: String
- password: String
+ Person()
+ Person(String, String, String, String)
+ setName(String): void
+ setHpNo(String): void
+ setID(String): void
+ setPW(String): void
+ getName(): String
+ getHpNo(): String
+ getID(): String
+ getPW(): String
+abstract displayInfo(): String
+abstract searchBook(ArrayList<Book>, SearchBookDialog): String
+addNewJournal(ArrayList<Book>, AddJournalPage): ArrayList<Book>
+addNewNovel(ArrayList<Book>, AddNovelPage): ArrayList<Book>
+addNewReferenceBook(ArrayList<Book>, AddReferenceBookPage): ArrayList<Book>
+deleteBook(ArrayList<Book>, BookDialog): ArrayList<Book>
+returnBook(ArrayList<Book>, BookDialog): ArrayList<Book>
+borrowBook(ArrayList<Book>, BookDialog): ArrayList<Book>
+displayBorrowed(DisplayWindow): void
+displayBorrowList(): String
+bookingIndividualRoom(ArrayList<StudyRoom>, IndividualRoomPage): ArrayList<StudyRoom>
+bookingGroupRoom(ArrayList<StudyRoom>, GroupRoomPage): ArrayList<StudyRoom>
+displayBookingList(DisplayWindow): void
Staff
- position: String
+ Staff()
+ Staff(String, String, String, String, String)
+ setPosition(String): void
+ getPosition(): String
+ getPosition(): String
+addNewJournal(ArrayList<Book>, AddJournalPage): ArrayList<Book>
+addNewNovel(ArrayList<Book>, AddNovelPage): ArrayList<Book>
+addNewReferenceBook(ArrayList<Book>, AddReferenceBookPage): ArrayList<Book>
+deleteBook(ArrayList<Book>, BookDialog): ArrayList<Book>
Student
- borrowList: Vector<Book>
- studyRoom: <StudyRoom>
- limit = 5: int
- id1: String
- id2: String
- id3: String
+ Student()
+ Student(String, String, String, String)
+returnBook(ArrayList<Book>, BookDialog): ArrayList<Book>
+borrowBook(ArrayList<Book>, BookDialog): ArrayList<Book>
+displayBookingList(DisplayWindow): void
+bookingGroupRoom(ArrayList<StudyRoom>, GroupRoomPage): ArrayList<StudyRoom>
+bookingIndividualRoom(ArrayList<StudyRoom>, IndividualRoomPage): ArrayList<StudyRoom>
+displayBorrowList(): String
+displayBorrowed(DisplayWindow): void

Diagram 4 show the Person class UML.

❖ Overall UML Diagram for GUI

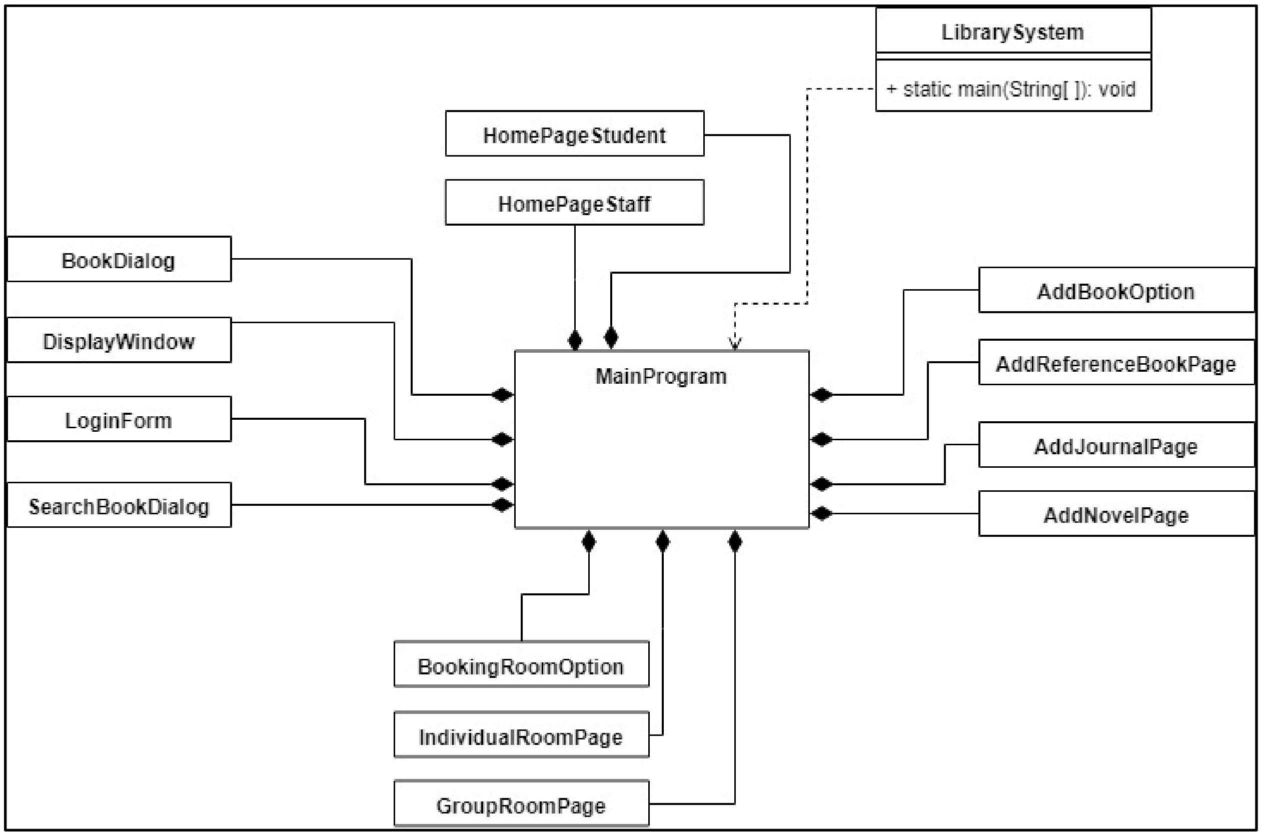


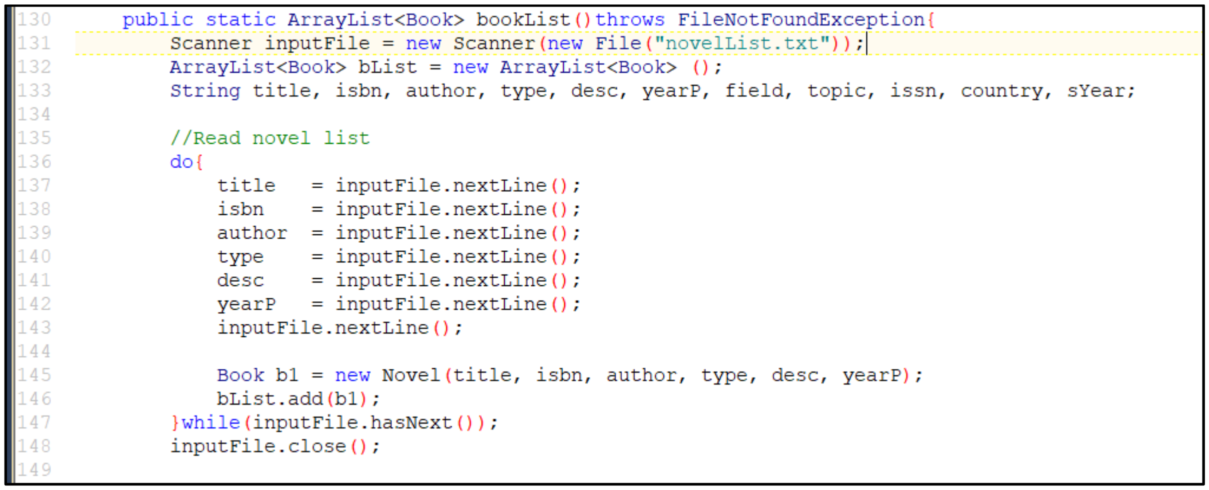
Diagram 5 show the overall GUI for our system.

C. The source code that Highlight concepts used

1. File class

In the system, the concept of file class has been applied to read the data of books. First, we had prepared three text file which contains the details of novel, journal and reference book separately. Then, the file had been read in the **library system class** and store in an ArrayList- bookList.

Then, the **Scanner class** had been used to open the text file (novelList.txt, journal.txt, referenceBookList.txt). In order to detect the end of the file, **while...hasNext()** loop had been used to read through the file data. After that, the input file needs to close.



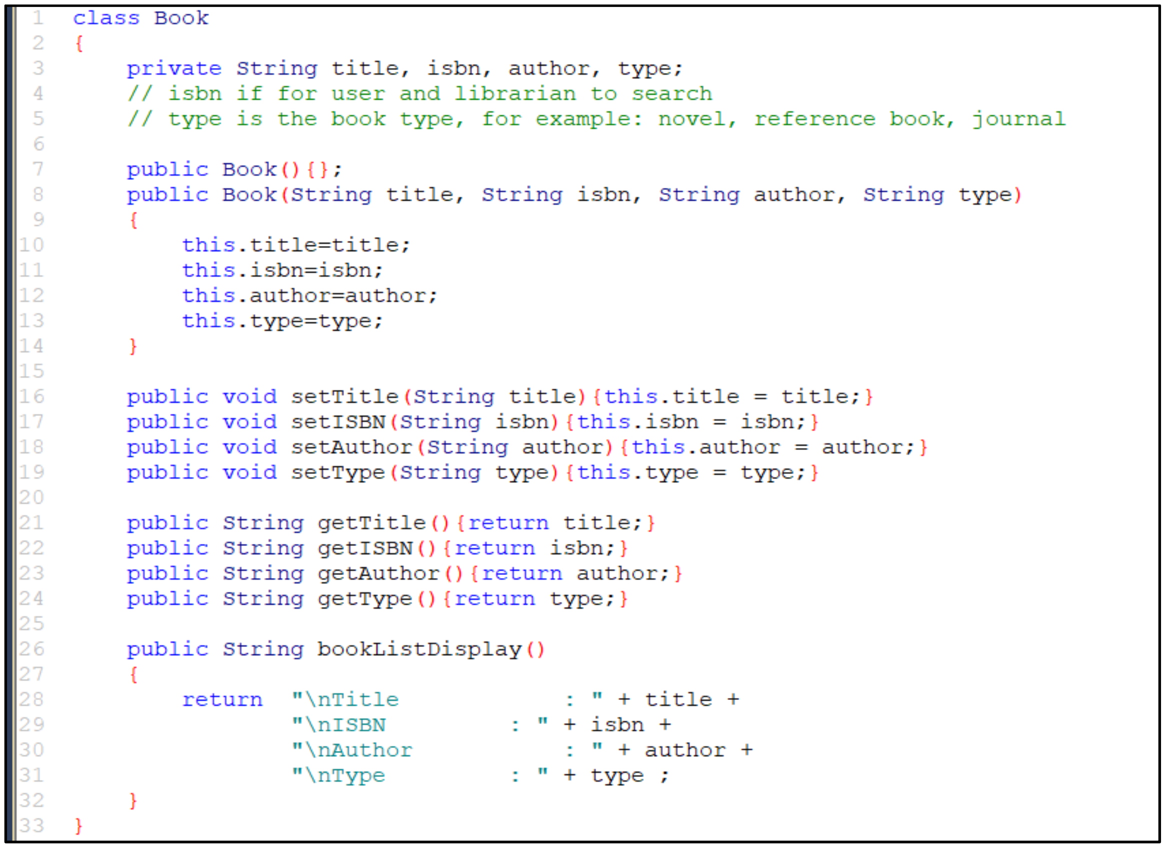
```
    public static ArrayList<Book> bookList()throws FileNotFoundException{
        Scanner inputFile = new Scanner(new File("novelList.txt"));
        ArrayList<Book> bList = new ArrayList<Book> ();
        String title, isbn, author, type, desc, yearP, field, topic, issn, country, sYear;

        //Read novel list
        do{
            title   = inputFile.nextLine();
            isbn    = inputFile.nextLine();
            author  = inputFile.nextLine();
            type    = inputFile.nextLine();
            desc    = inputFile.nextLine();
            yearP   = inputFile.nextLine();
            inputFile.nextLine();

            Book b1 = new Novel(title, isbn, author, type, desc, yearP);
            bList.add(b1);
        }while(inputFile.hasNext());
        inputFile.close();
```

Diagram 6 show the application of concept file class. Two more text file ("journal.txt", "referenceBookList.txt") had been read by using the same concept and method as well.

2. Encapsulation and Data Hiding

Encapsulation is a mechanism of wrapping the data (variable) and code action on the data (method) together as a single unit. In encapsulation, the variable of a class will be hidden from other classes and can be accessed only through the method of their current class. Therefore, it is also known as data hiding. Thus, **private** keyword had been used for those variables that decided to hide, and **public** keyword had been used in the accessor and mutator method to view and modify the variable value.

This concept we had applied in all the classes so that a class can have total control over what is stored in its field. For example, in Book class, variable title, ISBN, author and type has set as private and all the accessors and mutators for each variable had been set as public. The other classes can only access to those private variables through the public methods.



```java
class Book
{
    private String title, isbn, author, type;
    // isbn if for user and librarian to search
    // type is the book type, for example: novel, reference book, journal

    public Book(){};
    public Book(String title, String isbn, String author, String type)
    {
        this.title=title;
        this.isbn=isbn;
        this.author=author;
        this.type=type;
    }

    public void setTitle(String title){this.title = title;}
    public void setISBN(String isbn){this.isbn = isbn;}
    public void setAuthor(String author){this.author = author;}
    public void setType(String type){this.type = type;}

    public String getTitle(){return title;}
    public String getISBN(){return isbn;}
    public String getAuthor(){return author;}
    public String getType(){return type;}

    public String bookListDisplay()
    {
        return  "\nTitle            : " + title +
                "\nISBN          : " + isbn +
                "\nAuthor           : " + author +
                "\nType          : " + type ;
    }
}
```

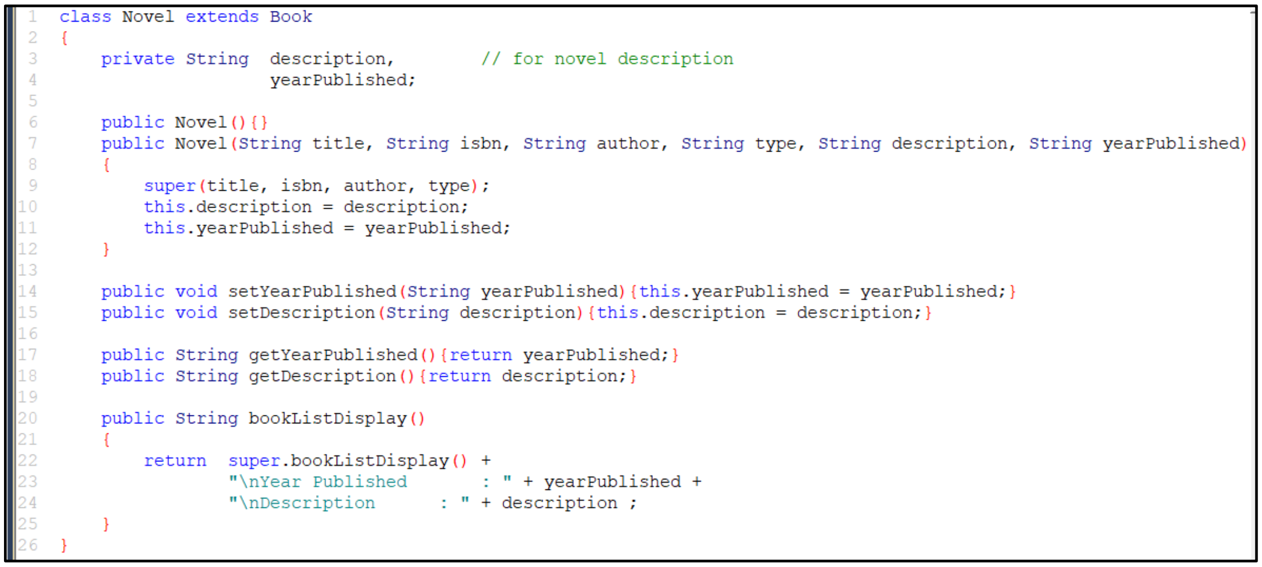
Diagram 7 show the application of concept encapsulation and data hiding in Book class.

3. <u>Class Relationship</u>

- Inheritance

  Inheritance is a mechanism in which one object acquires all the properties and behaviours of a parent object. The idea is that a new class which known as child class is created upon existing class, parent class. The child class is inherited from the parent class which means that the child class can use the method and fields of the parent class, and also, child class is allowed to have it owns new methods and fields. By applying the concept of inheritance, all the information is made manageable in a hierarchical order and enable the use of the overriding method as well. The **extends** keyword is used to indicate the the class is derived from the other existing class, for example, **class Novel extends Book{}**.

This concept we had applied in **Person class** and **Book class**. In our system, **Novel class**, **Journal class** and **ReferenceBook class** are inherit from **Book class**. Therefore, Book class contain the general variable, for example, title, ISBN, author and type. All of these variables will be inherited to the child classes. Book class also contain a method called bookListDisplay(). Besides the parent class's variable, every child classes have their own variable respectively. For example, the Novel class contains description and yearPublished variables. In child class constructor, **super()** keyword had used to refer to the constructor of the parent class. Besides, notice that in the **bookListDisplay()** of the child class, the first statement, **super.bookListDisplay()** is used to call the bookListDisplay() in the parent class.

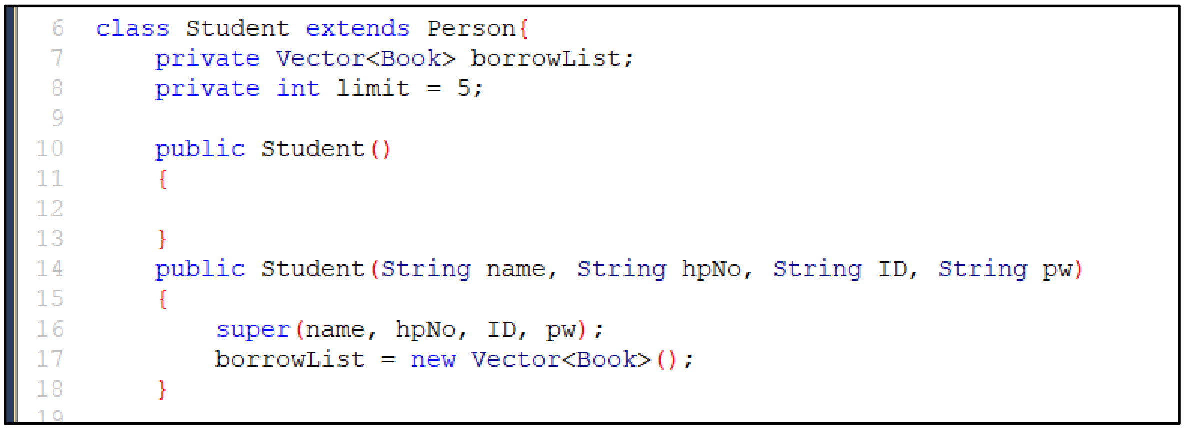


```
class Novel extends Book
{
    private String  description,        // for novel description
                    yearPublished;

    public Novel(){}
    public Novel(String title, String isbn, String author, String type, String description, String yearPublished)
    {
        super(title, isbn, author, type);
        this.description = description;
        this.yearPublished = yearPublished;
    }

    public void setYearPublished(String yearPublished){this.yearPublished = yearPublished;}
    public void setDescription(String description){this.description = description;}

    public String getYearPublished(){return yearPublished;}
    public String getDescription(){return description;}

    public String bookListDisplay()
    {
        return  super.bookListDisplay() +
                "\nYear Published       : " + yearPublished +
                "\nDescription      : " + description ;
    }
}
```

Diagram 8 show the application of the concept of inheritance relationship in Book class and Novel class.

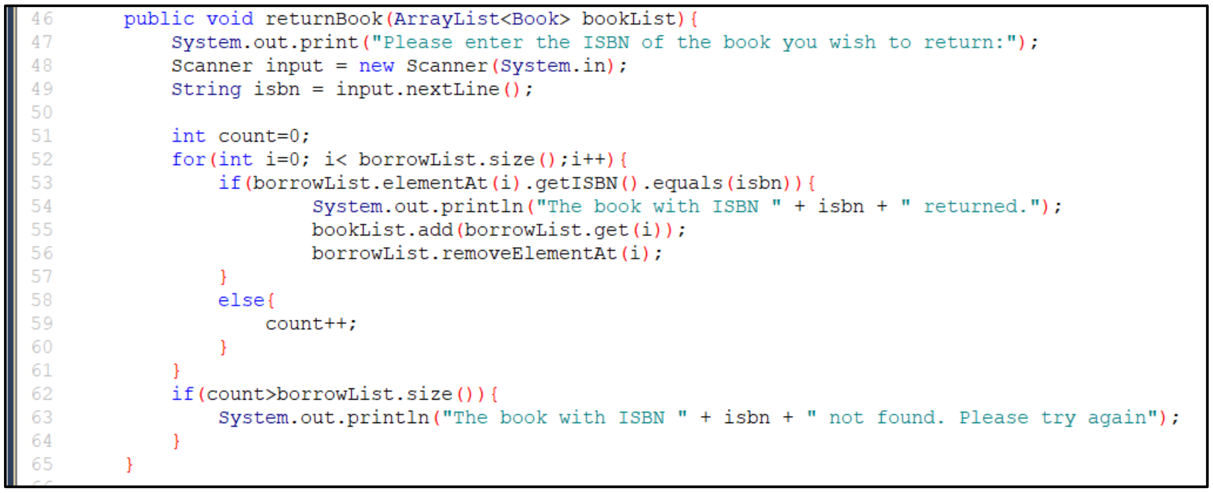
❖ Aggregation

Aggregation is a relationship between two classes that is best described as a "has-a" relationship. It contains a reference to another class. In our system, we had applied the aggregation concept in between **Student class** and **Book class**. Student class has aggregated the book class. Therefore, Student class can access to the method in Book class through Book class object. In the attribute declaration part, we can see that one of the attributes of Student class is borrowList which is the object of Book class. Thus, for the methods in Student class, like **returnBook()**, we can access to the methods in Book class as well, for example, **borrowList.elementAt(i)getISBN()**.



```java
class Student extends Person{
    private Vector<Book> borrowList;
    private int limit = 5;

    public Student()
    {

    }
    public Student(String name, String hpNo, String ID, String pw)
    {
        super(name, hpNo, ID, pw);
        borrowList = new Vector<Book>();
    }
```

Diagram 9 show the application of the concept of the aggregation relationship - Attribute declaration



```java
    public void returnBook(ArrayList<Book> bookList){
        System.out.print("Please enter the ISBN of the book you wish to return:");
        Scanner input = new Scanner(System.in);
        String isbn = input.nextLine();

        int count=0;
        for(int i=0; i< borrowList.size();i++){
            if(borrowList.elementAt(i).getISBN().equals(isbn)){
                    System.out.println("The book with ISBN " + isbn + " returned.");
                    bookList.add(borrowList.get(i));
                    borrowList.removeElementAt(i);
            }
            else{
                count++;
            }
        }
        if(count>borrowList.size()){
            System.out.println("The book with ISBN " + isbn + " not found. Please try again");
        }
    }
```

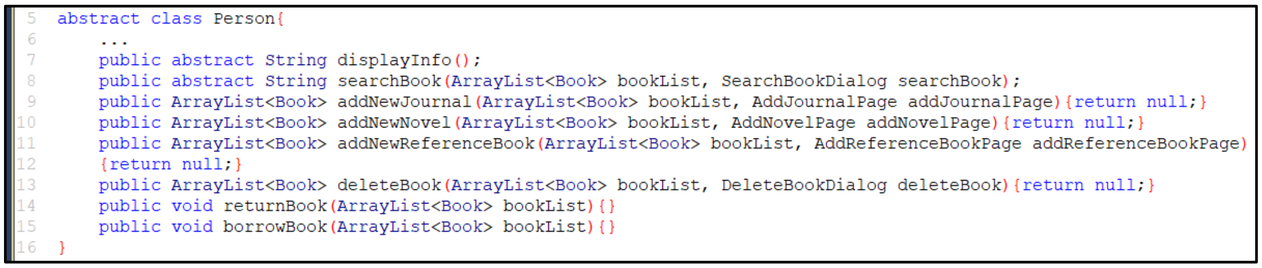
Diagram 10 show the application of the concept of the aggregation relationship - method used.

4. Polymorphism

❖ Abstract

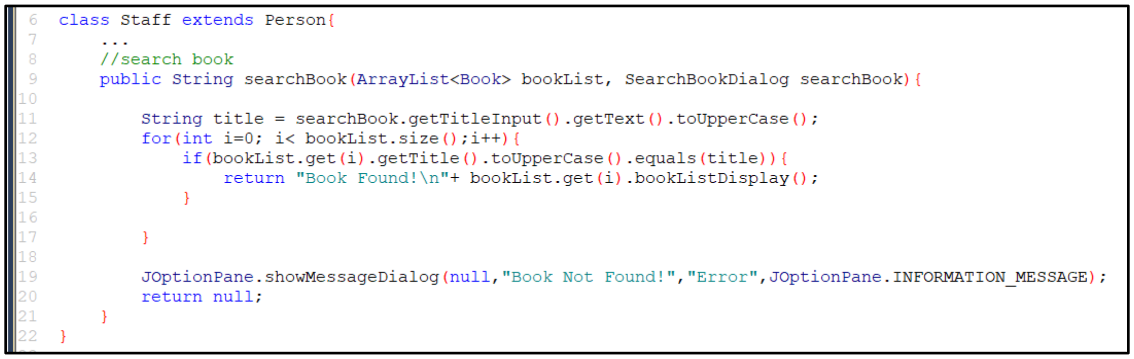
An abstract class is a superclass that cannot be instantiated and is used to state or define general characteristics. Keyword **abstract** is used to indicate abstract class. Besides, since not every method in the abstract class is the abstract method, for those abstract method, the abstract keyword needs to include in the method declaration and the method must have no body and must be overridden in a subclass.

We had applied the concept of polymorphism in Parent class and its child classes in our system. Since we want to create a superclass - **Person class** - that only defines a generalization form that will be shared by all of its subclasses - **Student class** and **Staff class**- leaving it to each subclass to fill in the details. The reasons that we implement abstract concept not only it reduces the complexity of viewing of the code, but also increase the reusability of code and security of an application as only important details are provided to the user. For example, in Person class, the method **searchBook()** is declared as abstract and have no body. On the other hand, in Staff class which extends from the Person class, has searchBook() as well with body.



```java
abstract class Person{
    ...
    public abstract String displayInfo();
    public abstract String searchBook(ArrayList<Book> bookList, SearchBookDialog searchBook);
    public ArrayList<Book> addNewJournal(ArrayList<Book> bookList, AddJournalPage addJournalPage){return null;}
    public ArrayList<Book> addNewNovel(ArrayList<Book> bookList, AddNovelPage addNovelPage){return null;}
    public ArrayList<Book> addNewReferenceBook(ArrayList<Book> bookList, AddReferenceBookPage addReferenceBookPage)
    {return null;}
    public ArrayList<Book> deleteBook(ArrayList<Book> bookList, DeleteBookDialog deleteBook){return null;}
    public void returnBook(ArrayList<Book> bookList){}
    public void borrowBook(ArrayList<Book> bookList){}
}
```

Diagram 11 show application of concept polymorphism in the parent class.

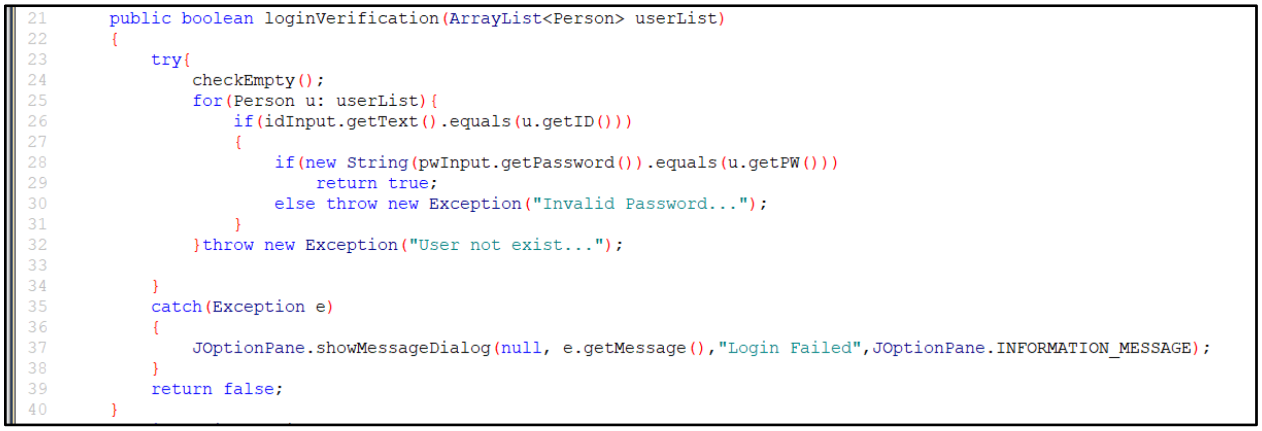


```java
class Staff extends Person{
    ...
    //search book
    public String searchBook(ArrayList<Book> bookList, SearchBookDialog searchBook){

        String title = searchBook.getTitleInput().getText().toUpperCase();
        for(int i=0; i< bookList.size();i++){
            if(bookList.get(i).getTitle().toUpperCase().equals(title)){
                return "Book Found!\n"+ bookList.get(i).bookListDisplay();
            }

        }

        JOptionPane.showMessageDialog(null,"Book Not Found!","Error",JOptionPane.INFORMATION_MESSAGE);
        return null;
    }
}
```

Diagram 12 show application of concept polymorphism in the child class.

5. Exception Handling

In our system, we had applied the exception handling concept in the **loginVerification()** part. In order to access the services, users had to login by using their id and password. If the user had input the wrong id or password, which means that exception occurs, it will execute the throw line. Then, the error message thrown will catch and display in message dialog.



```
public boolean loginVerification(ArrayList<Person> userList)
{
    try{
        checkEmpty();
        for(Person u: userList){
            if(idInput.getText().equals(u.getID()))
            {
                if(new String(pwInput.getPassword()).equals(u.getPW()))
                    return true;
                else throw new Exception("Invalid Password...");
            }
        }throw new Exception("User not exist...");

    }
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e.getMessage(),"Login Failed",JOptionPane.INFORMATION_MESSAGE);
    }
    return false;
}
```

Diagram 13 show application of concept exception handling

6. Vector & ArrayList

❖ Vector

Vector is similar to ArrayList which can store object dynamically, but the method to add or get element from Vector list is different with method in ArrayList. We implement Vector in the **borrowList** in the Student Class. In line 45, it shows that we add an Book object passed from **bookList** in main function into the **borrowList** by using Vector method, **addElement(obj).**

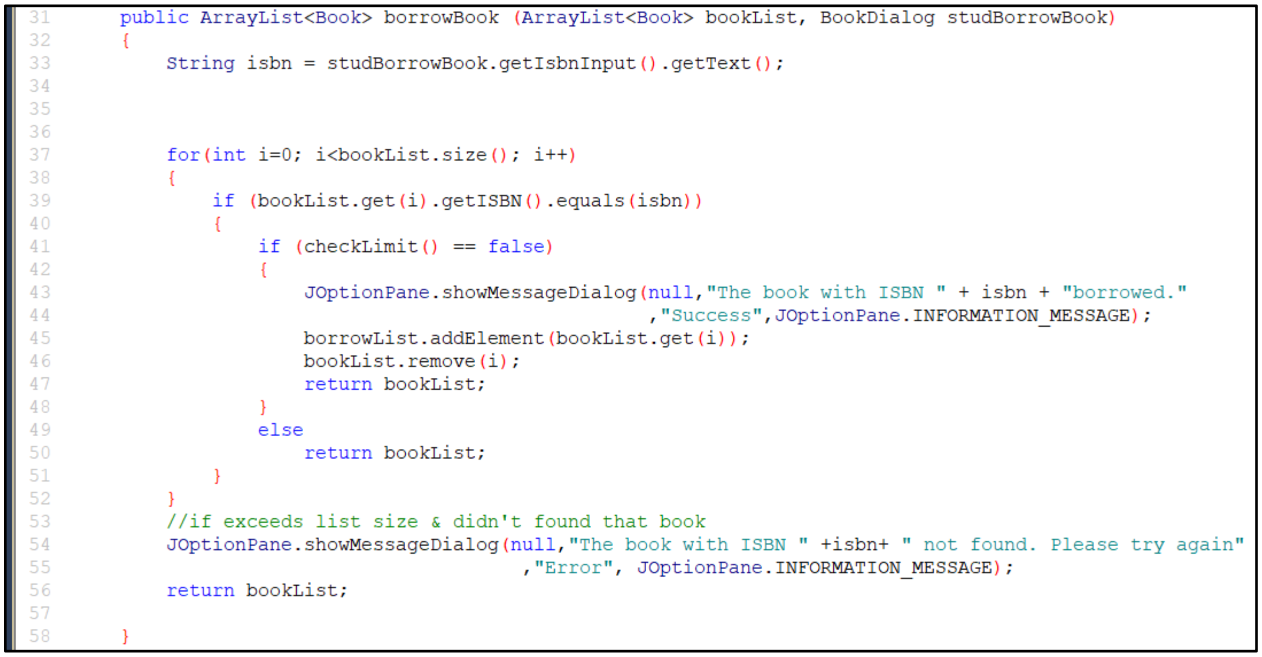
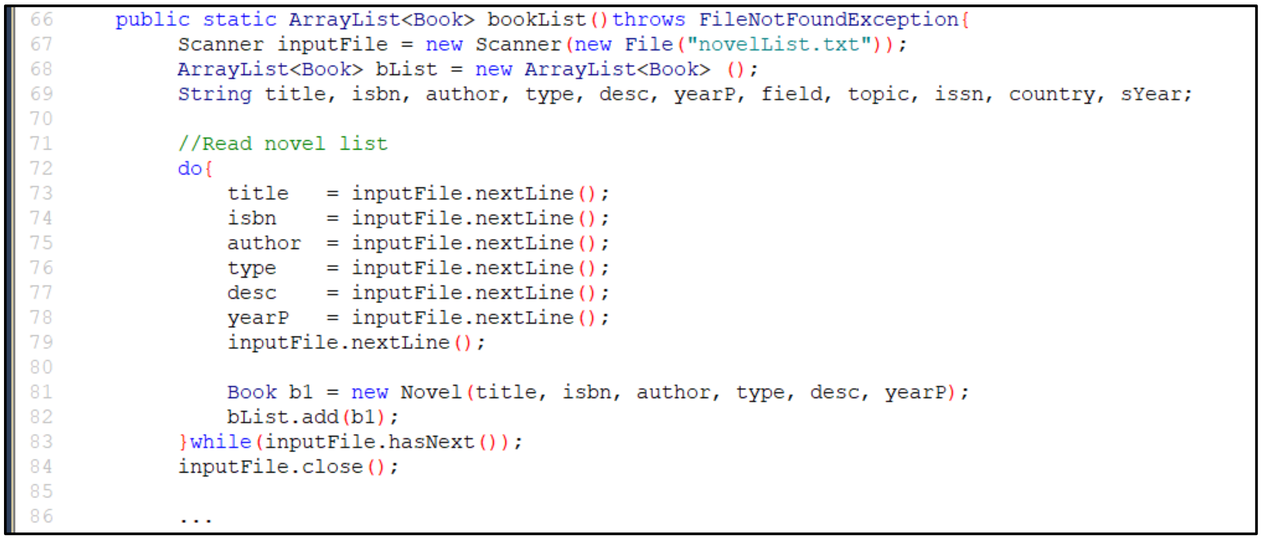

```java
    public ArrayList<Book> borrowBook (ArrayList<Book> bookList, BookDialog studBorrowBook)
    {
        String isbn = studBorrowBook.getIsbnInput().getText();



        for(int i=0; i<bookList.size(); i++)
        {
            if (bookList.get(i).getISBN().equals(isbn))
            {
                if (checkLimit() == false)
                {
                    JOptionPane.showMessageDialog(null,"The book with ISBN " + isbn + "borrowed."
                                                 ,"Success",JOptionPane.INFORMATION_MESSAGE);
                    borrowList.addElement(bookList.get(i));
                    bookList.remove(i);
                    return bookList;
                }
                else
                    return bookList;
            }
        }
        //if exceeds list size & didn't found that book
        JOptionPane.showMessageDialog(null,"The book with ISBN " +isbn+ " not found. Please try again"
                                     ,"Error", JOptionPane.INFORMATION_MESSAGE);
        return bookList;

    }
```

Diagram 14 show application of concept vector

❖ ArrayList

ArrayList is used to store dynamically sized collection of elements but contrary to Arrays that are fixed in size, ArrayList grows its size automatically when new elements are added to it. Due to this features, we had decided to implement ArrayList concept in **bookList**. This is because of the size for bookList is not fixed, staff are allowed to add new books or remove books from the list. Therefore, ArrayList can fulfil our requirement. Whenever there is a new book, its details can be recorded by using **add()** method.

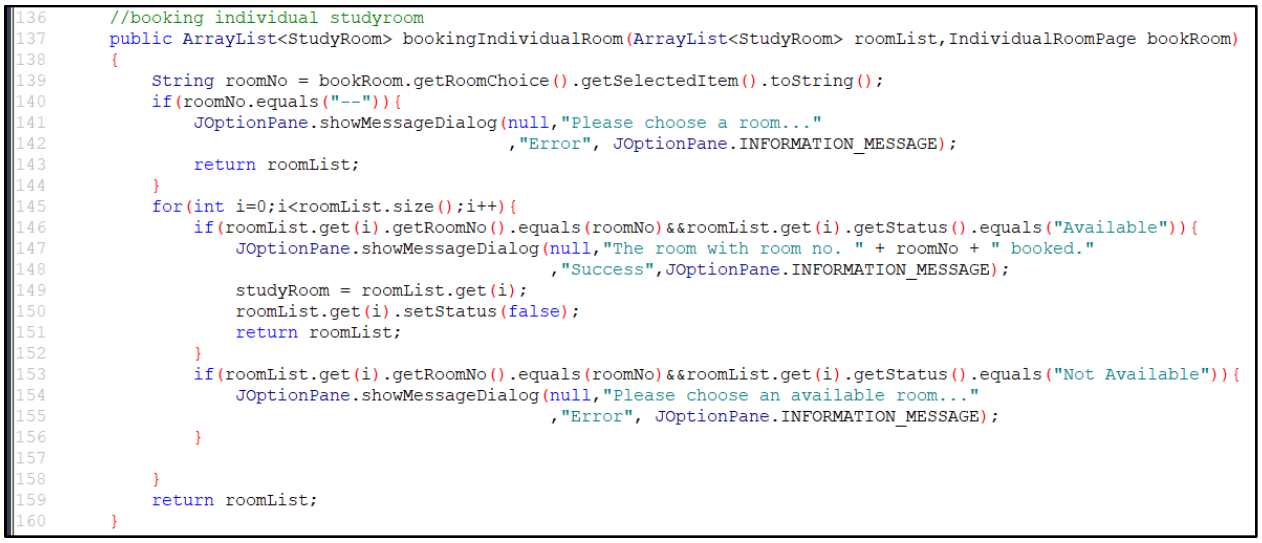


```
    public static ArrayList<Book> bookList()throws FileNotFoundException{
         Scanner inputFile = new Scanner(new File("novelList.txt"));
         ArrayList<Book> bList = new ArrayList<Book> ();
         String title, isbn, author, type, desc, yearP, field, topic, issn, country, sYear;

         //Read novel list
         do{
             title   = inputFile.nextLine();
             isbn    = inputFile.nextLine();
             author  = inputFile.nextLine();
             type    = inputFile.nextLine();
             desc    = inputFile.nextLine();
             yearP   = inputFile.nextLine();
             inputFile.nextLine();

             Book b1 = new Novel(title, isbn, author, type, desc, yearP);
             bList.add(b1);
         }while(inputFile.hasNext());
         inputFile.close();

         ...
```

Diagram 15 show the application of concept array list.

7. <u>GUI</u>

The input and output of our system is totally based on graphical user interface(GUI). We use the NetBeans IDE in our interface design. We have implemented Java libraries such as javax.swing and java.awt.event in our system. Basically, there is a Class for every page of the system interface, and the functions provided are also different. JTextField is always used to get input from user, and JTextArea is always used to display output for user. However, there are some conditions that information is displayed through other methods. For example, error message and success message is displayed to the users by message dialog.



```java
    //booking individual studyroom
    public ArrayList<StudyRoom> bookingIndividualRoom(ArrayList<StudyRoom> roomList,IndividualRoomPage bookRoom)
    {
        String roomNo = bookRoom.getRoomChoice().getSelectedItem().toString();
        if(roomNo.equals("--")){
            JOptionPane.showMessageDialog(null,"Please choose a room..."
                                          ,"Error", JOptionPane.INFORMATION_MESSAGE);
            return roomList;
        }
        for(int i=0;i<roomList.size();i++){
            if(roomList.get(i).getRoomNo().equals(roomNo)&&roomList.get(i).getStatus().equals("Available")){
                JOptionPane.showMessageDialog(null,"The room with room no. " + roomNo + " booked."
                                              ,"Success",JOptionPane.INFORMATION_MESSAGE);
                studyRoom = roomList.get(i);
                roomList.get(i).setStatus(false);
                return roomList;
            }
            if(roomList.get(i).getRoomNo().equals(roomNo)&&roomList.get(i).getStatus().equals("Not Available")){
                JOptionPane.showMessageDialog(null,"Please choose an available room..."
                                              ,"Error", JOptionPane.INFORMATION_MESSAGE);
            }

        }
        return roomList;
    }
```

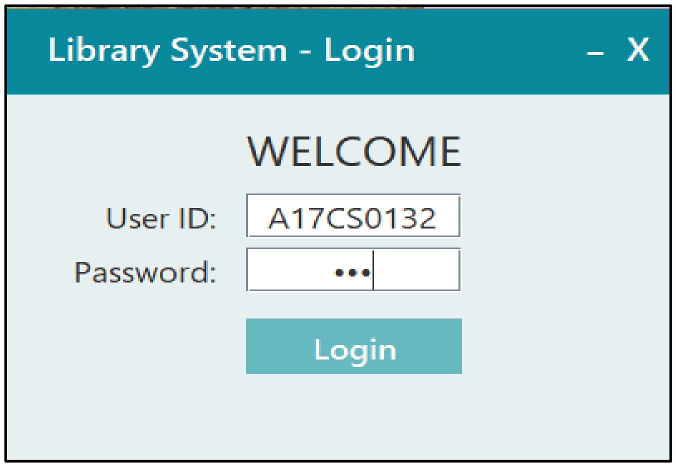
Diagram 16 show application of concept GUI

D. User manual (sample input/output for each task)

For the following input and output sample, it will display from both the student and staff perspective.

<u>Student Perspective</u>

1. First, the system will display an interface that prompt the user (in this case the user is student) to key-in their user ID and password as well.

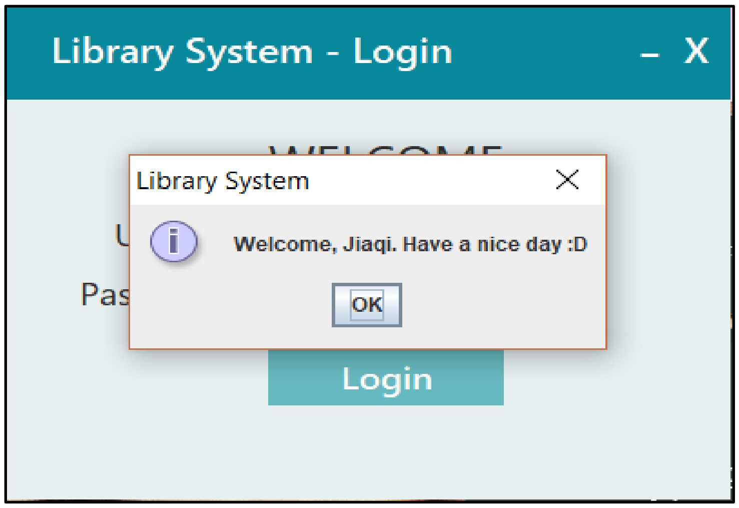


User ID: A17CS0132

Password: •••

Diagram 17 show the user key in his/her user ID and password.

2. Once the user had key in the correct user ID and password, the system will display a welcome interface and then direct the user to their respectives interface based on their User ID.



Welcome, Jiaqi. Have a nice day :D

Diagram 18 show the user had successfully login to the system.

3. After that, the system display the function that available for the student, which separated into two part- book operation and student operation.

<u>BOOK OPERATION</u>

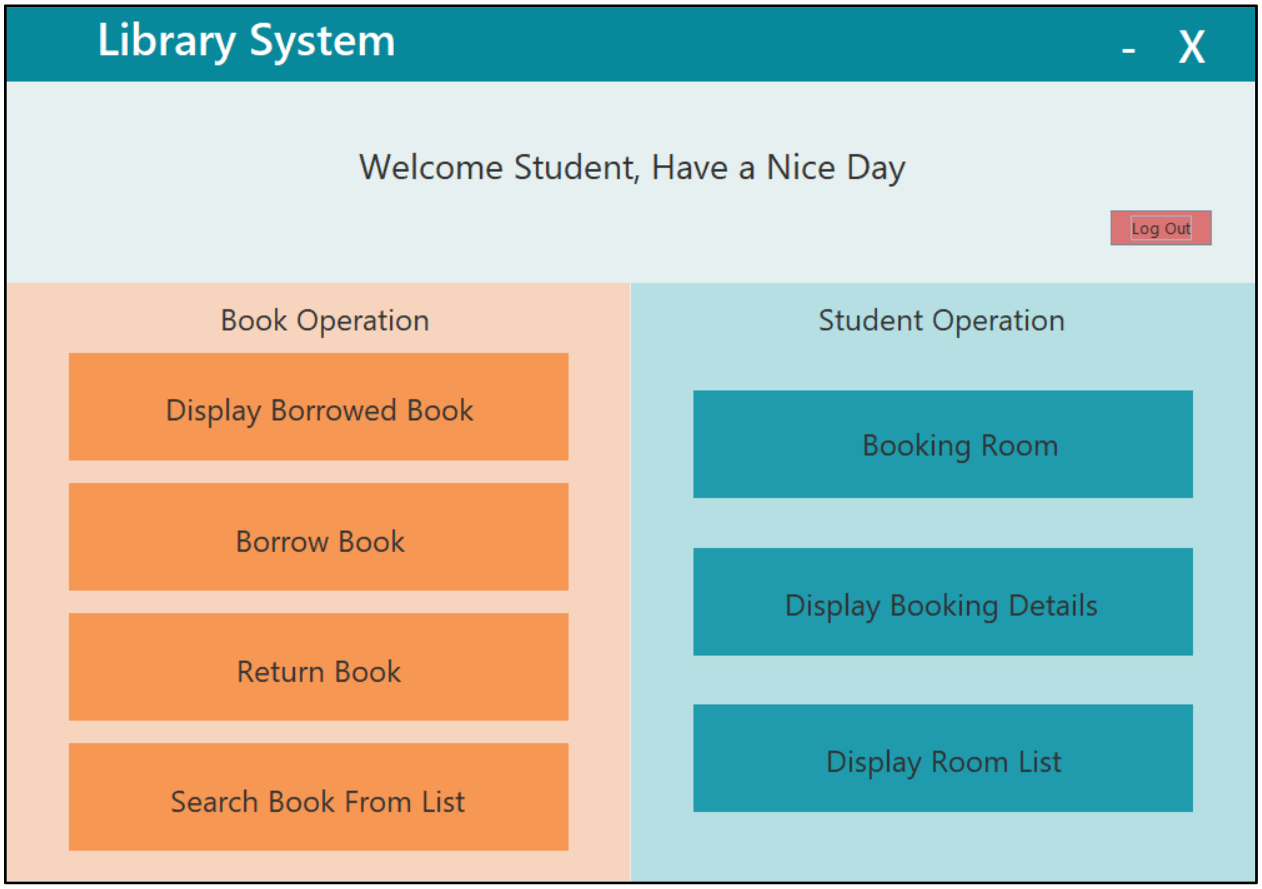


Diagram 19 show the student services interface.

4. When student click “**Borrow Book**”, system prompt the user to input ISBN for the book that he/she desired to borrow. After input, student need to click confirm and system will search for it.

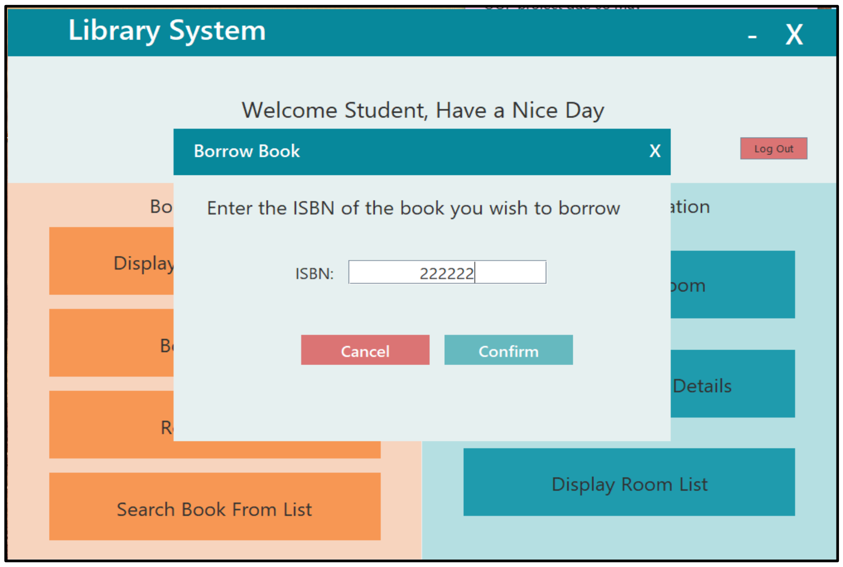


Diagram 20 show the student had input the ISBN of the first book he/ she wish to borrow.

5. Since each of the student can borrow up to 5 books, in this case, the student continue to select “**Borrow book**” function, and the system prompt user to input ISBN as well.

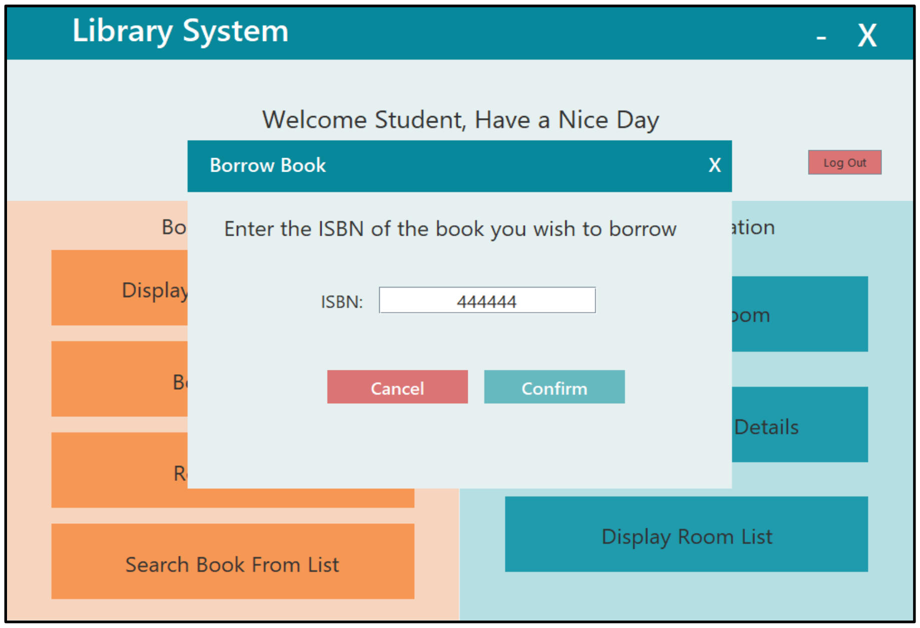


Diagram 21 show the student had input the ISBN of the second book he/ she wish to borrow.

6. If the book is available in library, system will display an "Successful borrowed" interface.

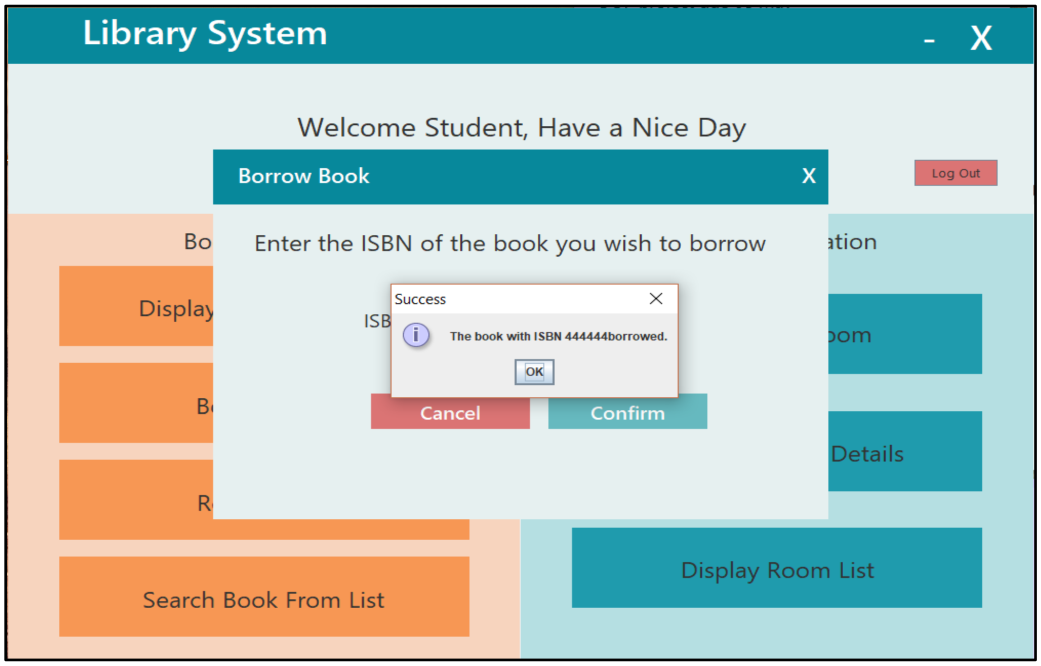
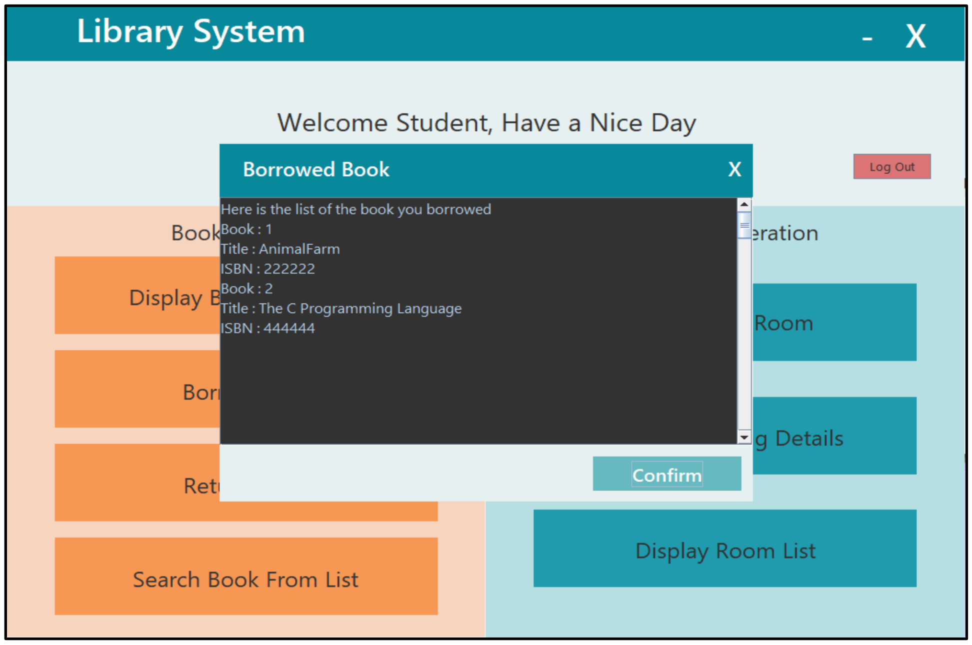


Diagram 22 show the system displayed successful borrowed interface.

7. After that, student select "**Displayed Borrowed Book**" to check the total books that he/she had borrowed.



```
Here is the list of the book you borrowed
Book : 1
Title : AnimalFarm
ISBN : 222222
Book : 2
Title : The C Programming Language
ISBN : 444444
```

Diagram 23 show the borrowed book list of the student.

8. Student select “**Search Book From List**” function to search and check the availability of the book that he/she desired by input the title of the book.

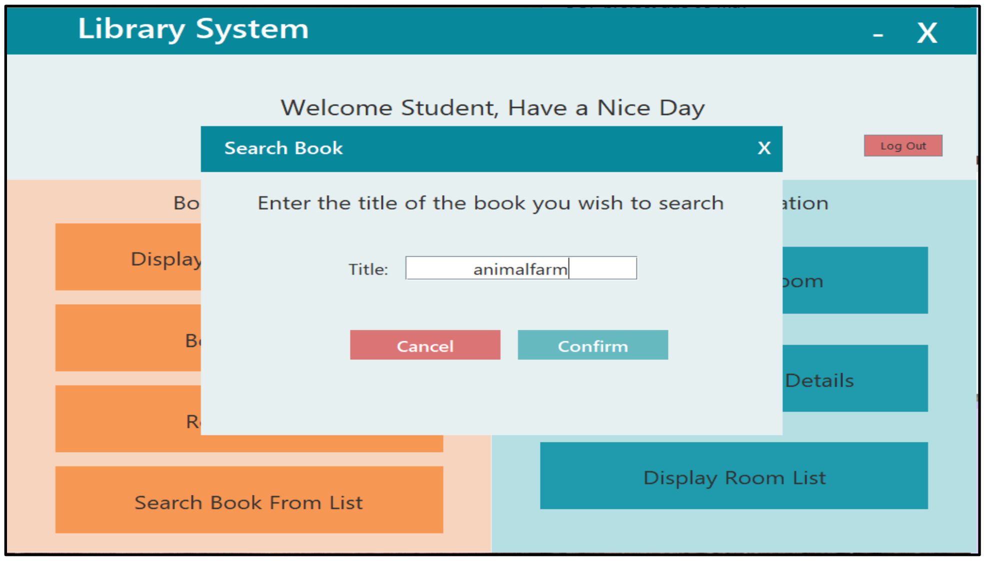
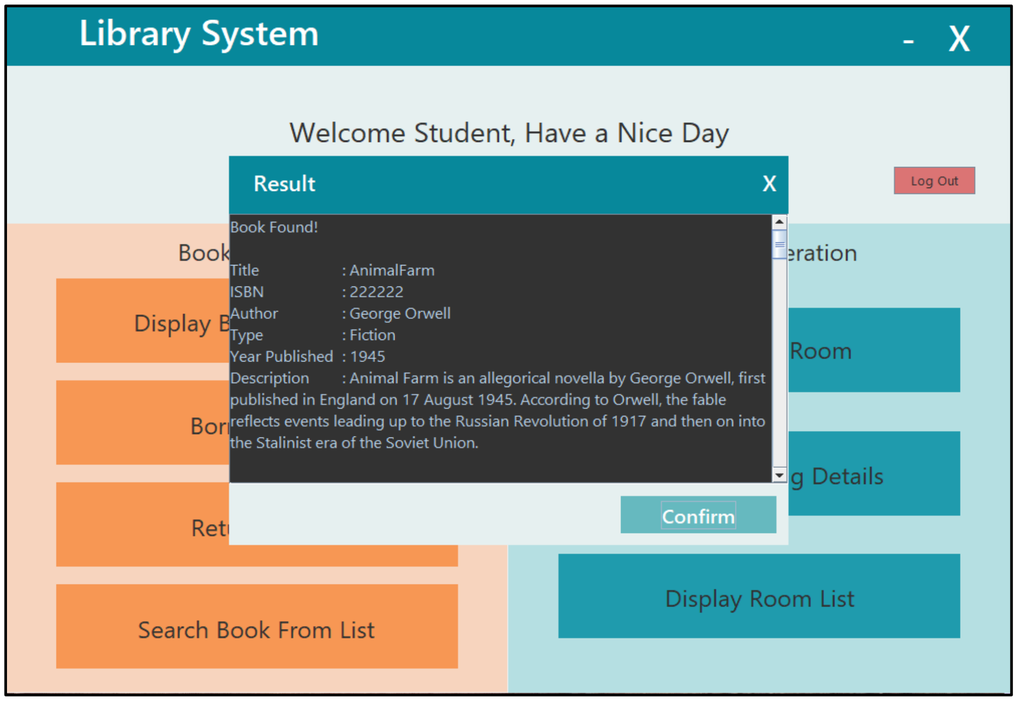


Diagram 24 show the user input the title of the book.

9. If the book that student search is currently available, the system will inform the student by displaying “**Book Found!**” and the details of the book, for example, title, ISBN, author, type, year of published and description of the book.



```
Book Found!

Title           : AnimalFarm
ISBN            : 222222
Author          : George Orwell
Type            : Fiction
Year Published  : 1945
Description     : Animal Farm is an allegorical novella by George Orwell, first
published in England on 17 August 1945. According to Orwell, the fable
reflects events leading up to the Russian Revolution of 1917 and then on into
the Stalinist era of the Soviet Union.
```

Diagram 25 show the system displayed the details of the book if it is found.

10. Student try to search another book by inserting the title of the book as well.

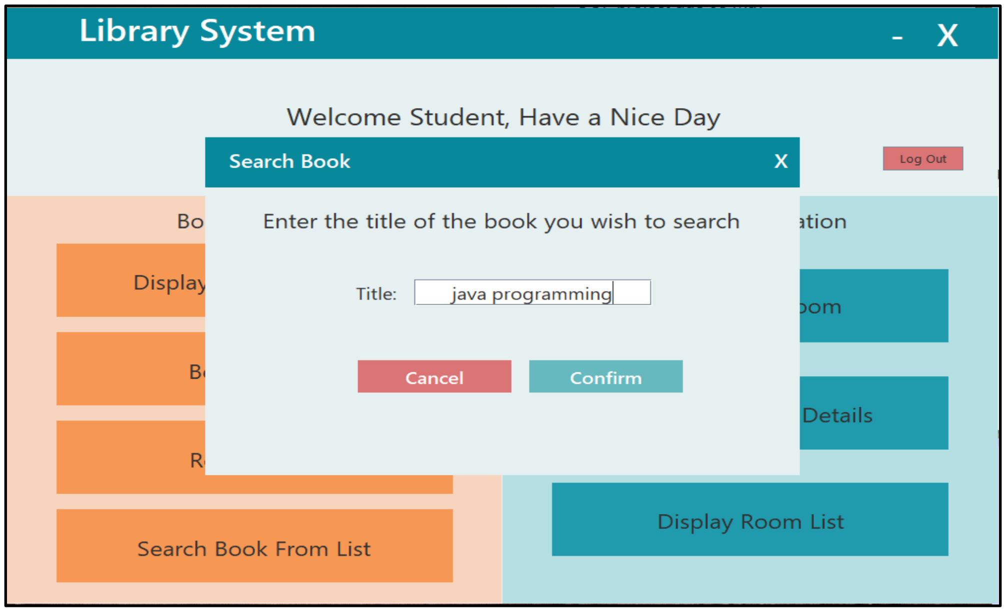
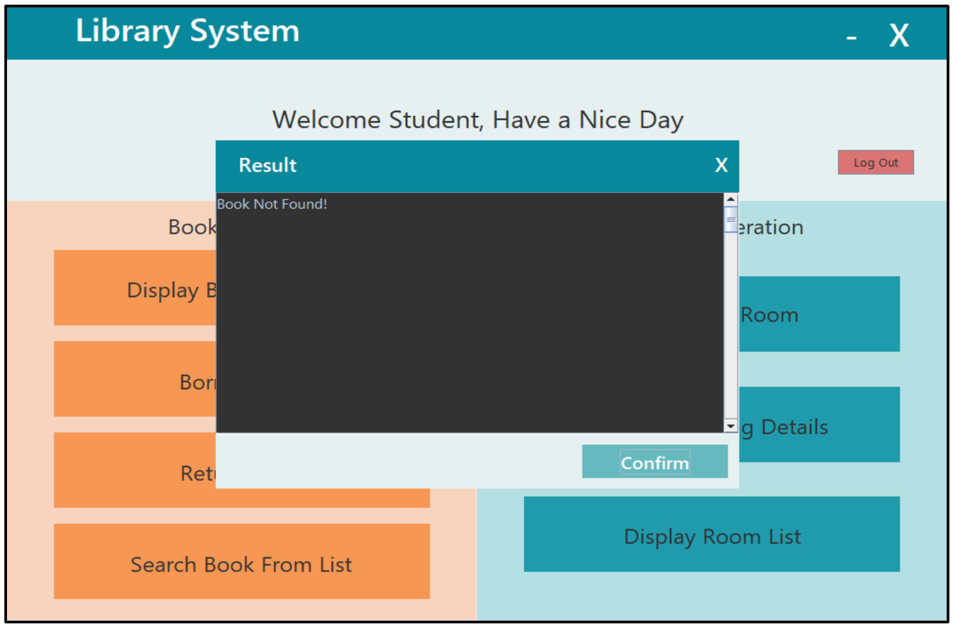


Diagram 26 show the student inserting the title of the book..

11. The book is not found or currently not available in library. Thus, the system displayed "**Book Not Found!**"



```
Book Not Found!
```

Diagram 27 show the system inform the student that the book is currently not available or not found in library.

12. Student choose “**Return Book**” to return the books that have been borrowed. The system will prompt user to enter the ISBN of the book.

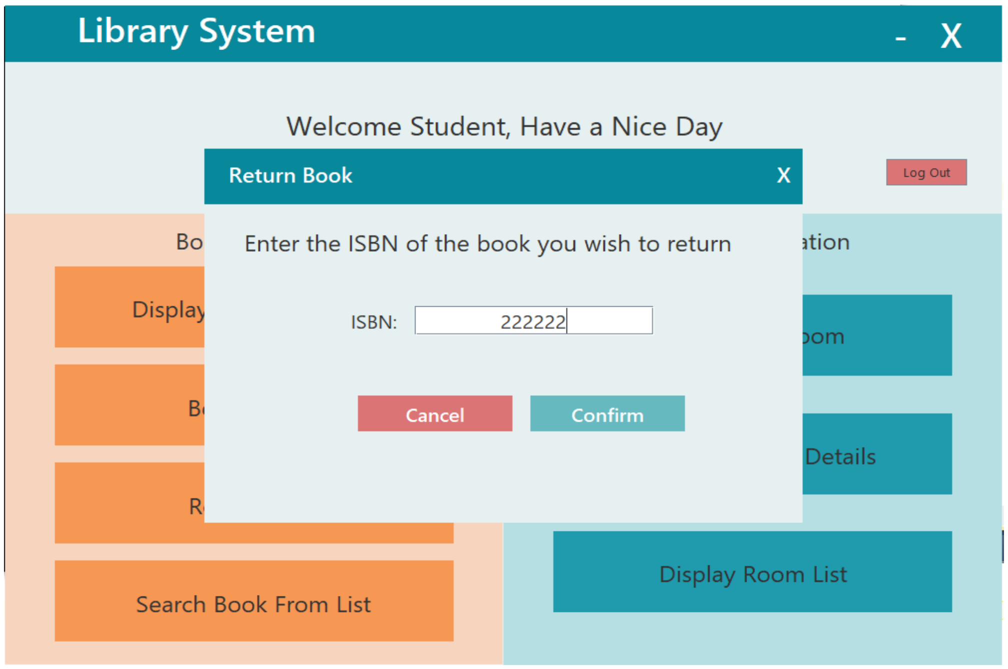
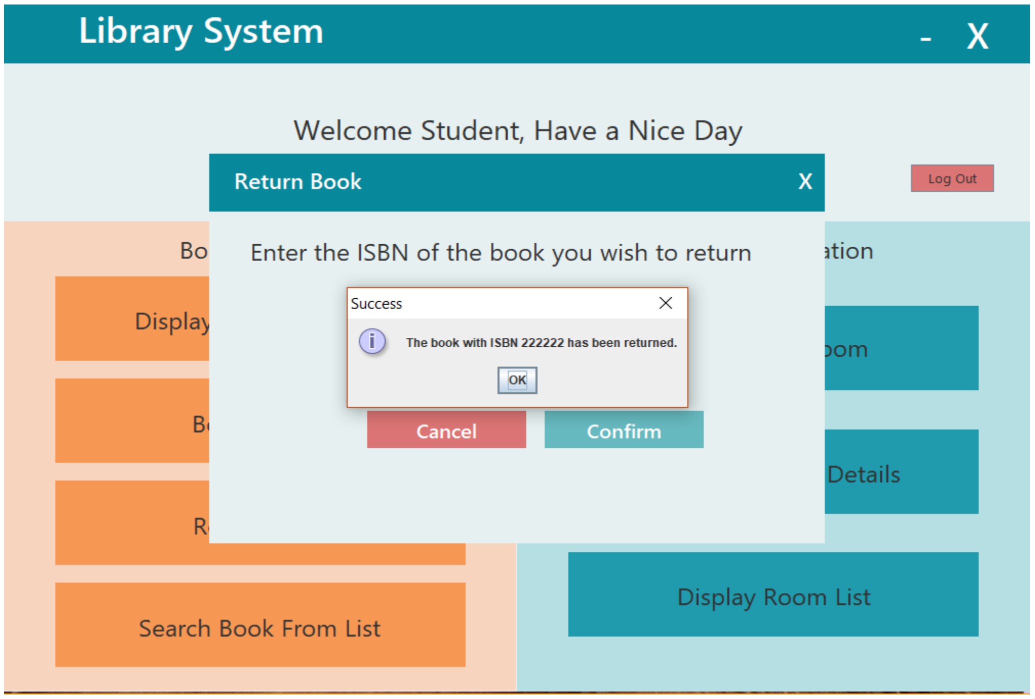


Diagram 28 show the student inserting the ISBN of the book that will be returned to the library

13. Since the book is in the student’s borrow list, so it can be successfully returned.

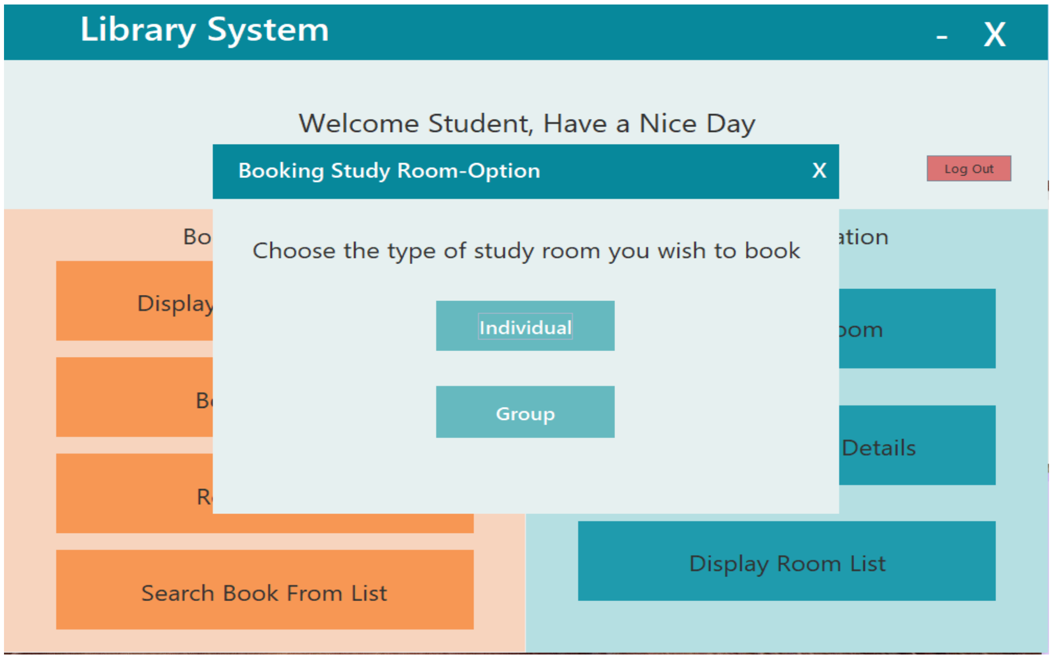


The book with ISBN 222222 has been returned.

Diagram 29 show the system informing the student that the book is successfully returned

STUDY ROOM OPERATION

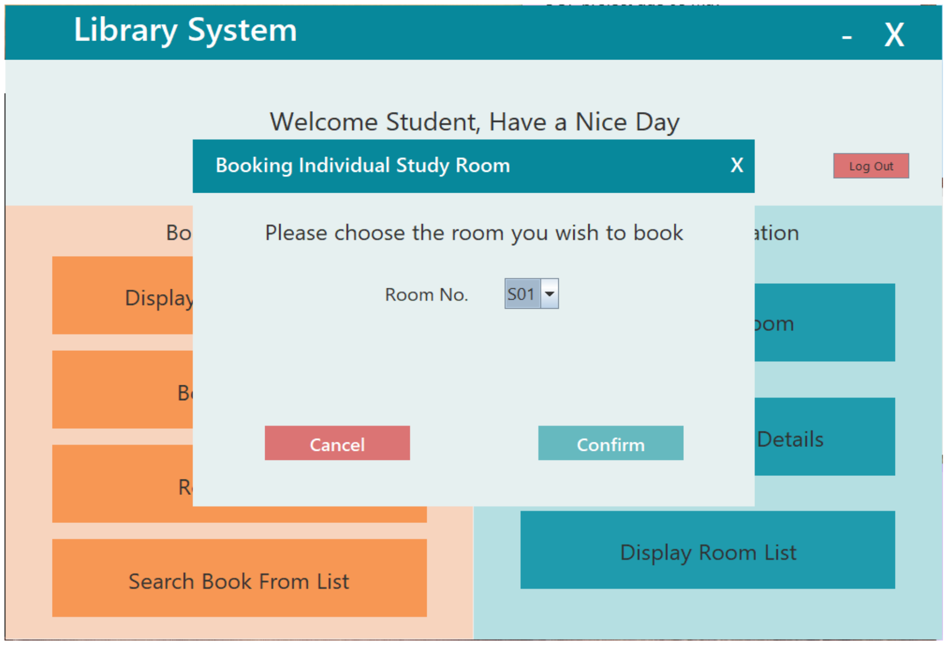
14. When a student wants to book a study room, he or she can choose "**Booking Room**" to book individual study room or group discussion room.



Choose the type of study room you wish to book

Diagram 30 show the system displays the options for the student to choose

15. The student choose "**Individual**", so the system will display the panel for individual room.



Please choose the room you wish to book

Room No. S01

Diagram 31 show the student choosing the individual room with roomNo S01

16. Since the room is available for now, the system will tell user the booking is success.

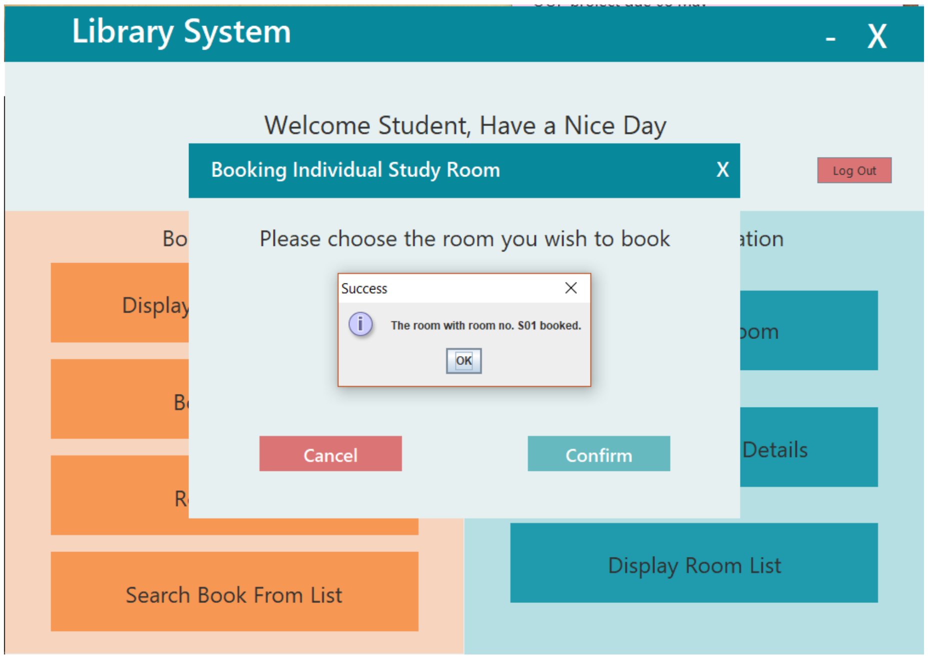
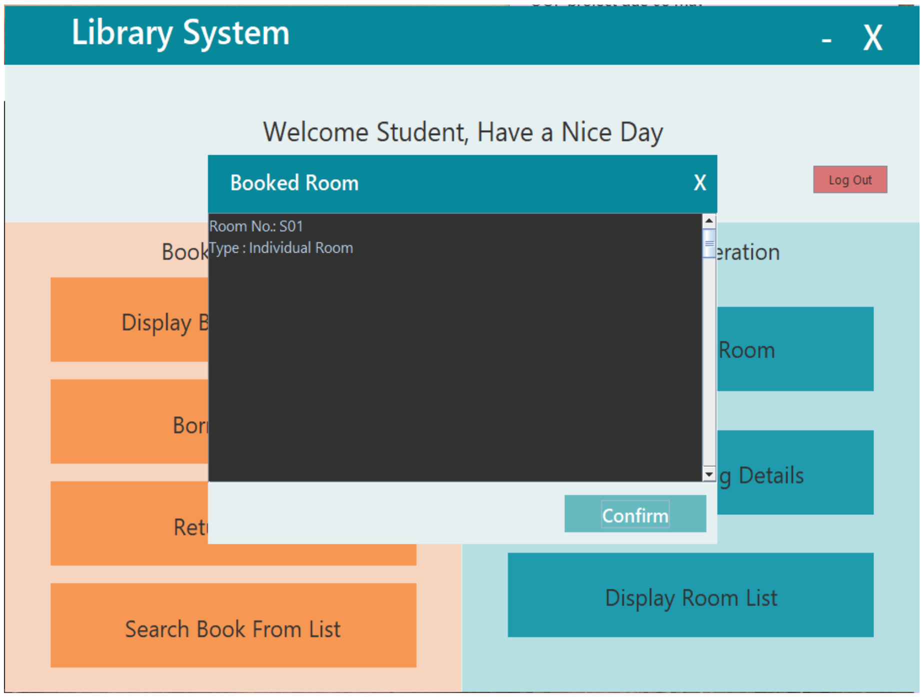

Library System
-
X
Welcome Student, Have a Nice Day
Booking Individual Study Room
X
Log Out
Bo
Please choose the room you wish to book
ation
Display
Success
The room with room no. S01 booked.
OK
oom
B
Details
Cancel
Confirm
R
Display Room List
Search Book From List

Diagram 32 show the system telling the student room S01 is successfully booked

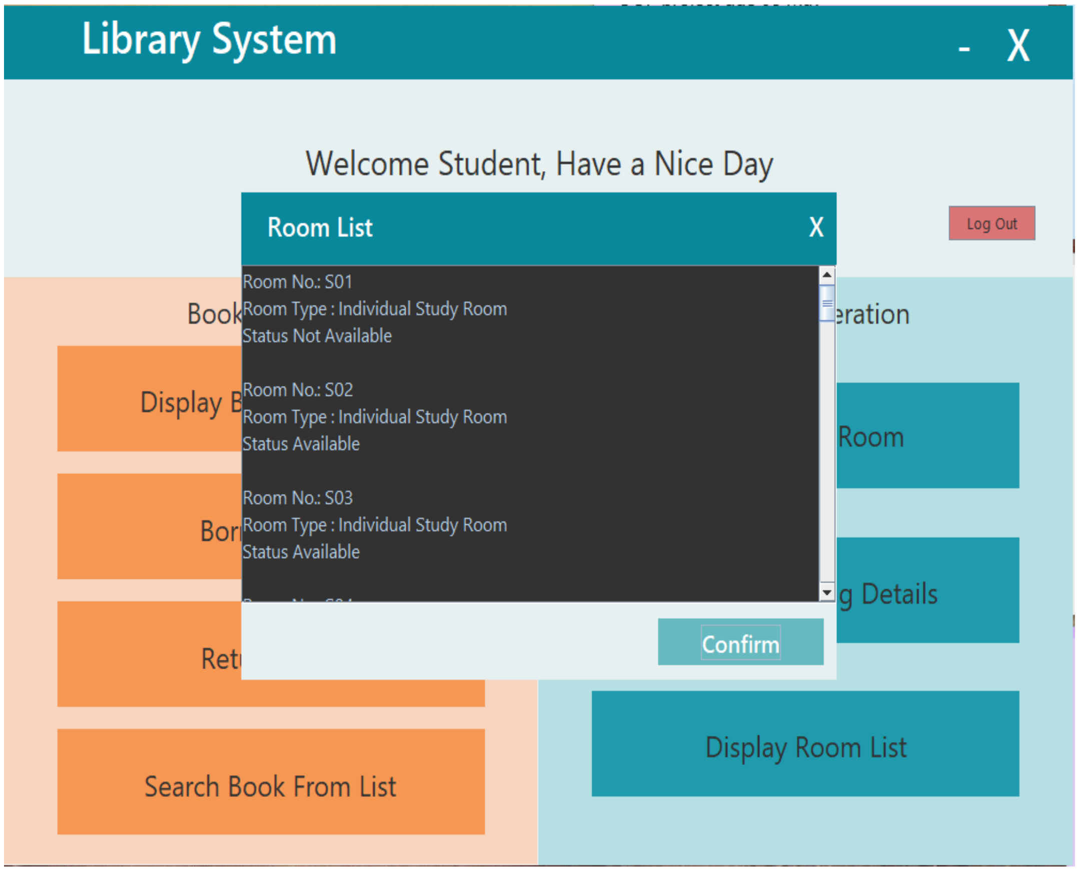
17. Then, the student can check the booking details by choosing "**Display Booking Details**" to see whether there is room booked and the booked room details.



```
Room No.: S01
Type : Individual Room
```

Diagram 33 show the system display the details of the room booked, S01

18. The student may browse the room list before or after booking a room by choosing "**Display Room List**". Since room S01 is booked, the status of the room will display as "**Not Available**".



Room No.: S01
Room Type : Individual Study Room
Status Not Available

Room No.: S02
Room Type : Individual Study Room
Status Available

Room No.: S03
Room Type : Individual Study Room
Status Available

Diagram 34 show the system display the list of study rooms, and the status of room S01 is Not Available

Staff Perspective

Next, for the following input and output sample, it will display from staff perspective.

1. First, staff needs to login through the same interface as student by inserting their ID and password.

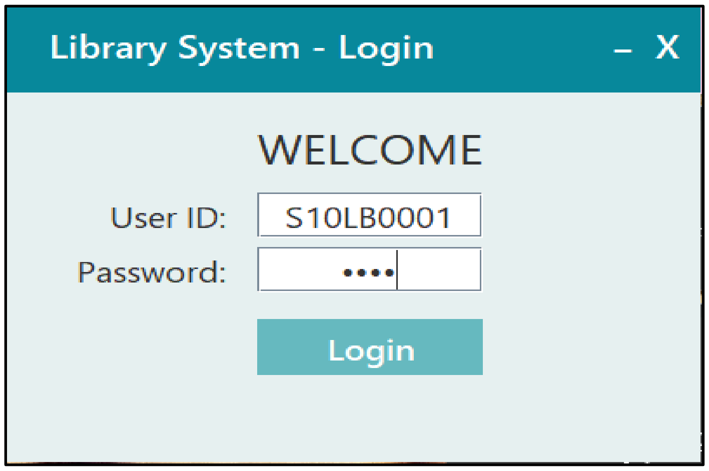
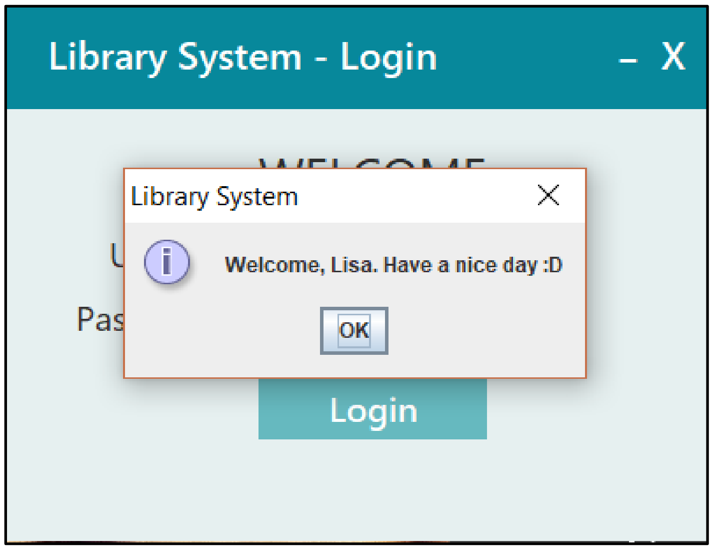


Diagram 35 show the staff input his/ her ID and password.

2. Once the staff had key in the correct user ID and password, the system will display a welcome interface and then direct the staff to their respectives interface based on their User ID.



Welcome, Lisa. Have a nice day :D

Diagram 36 show the staff had successfully login to the system.

3. After that, the system display the function that available for the staff, which separated into two part- book operation and student operation.

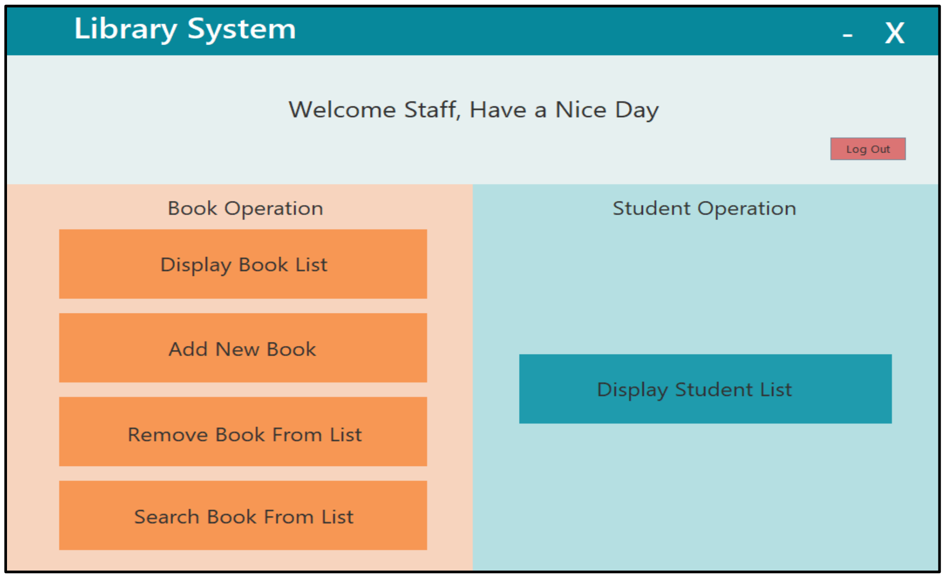
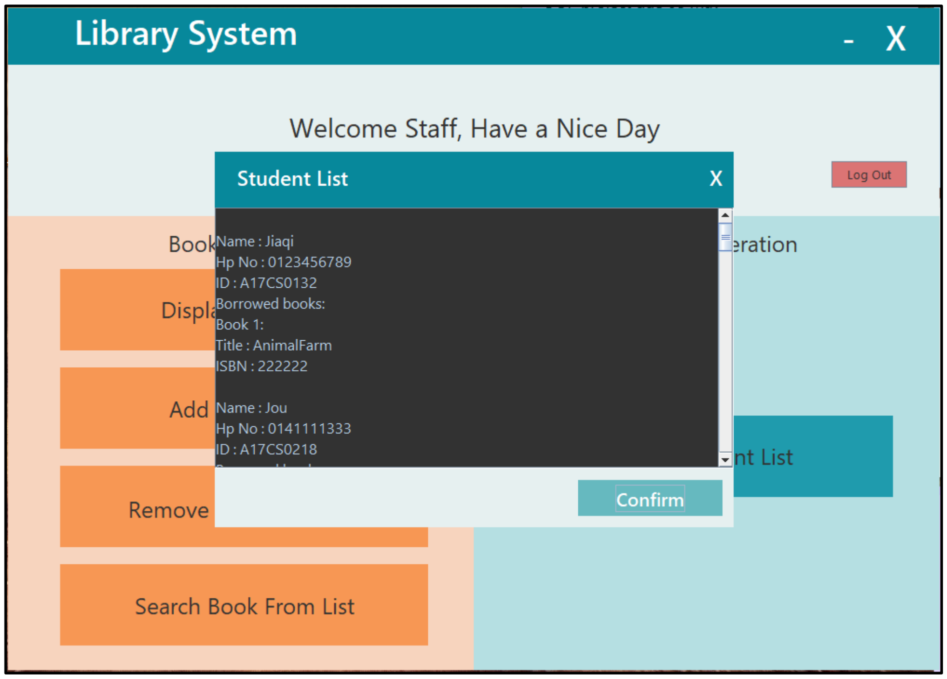


Diagram 37 show the student services interface.

4. Staff select "**Display Student List**" to check that the students that had made borrowed during that period. List display the details of the students and information of the borrowed book.



```
Name : Jiaqi
Hp No : 0123456789
ID : A17CS0132
Borrowed books:
Book 1:
Title : AnimalFarm
ISBN : 222222

Name : Jou
Hp No : 0141111333
ID : A17CS0218
```

Diagram 38 show the student list that had borrowed book.

5. Staff can update the details of new book in library from time to time by selecting "**Add New Book**", and system will display 3 types of book for the staff to choose, which is Journal, Novel and Reference Book.

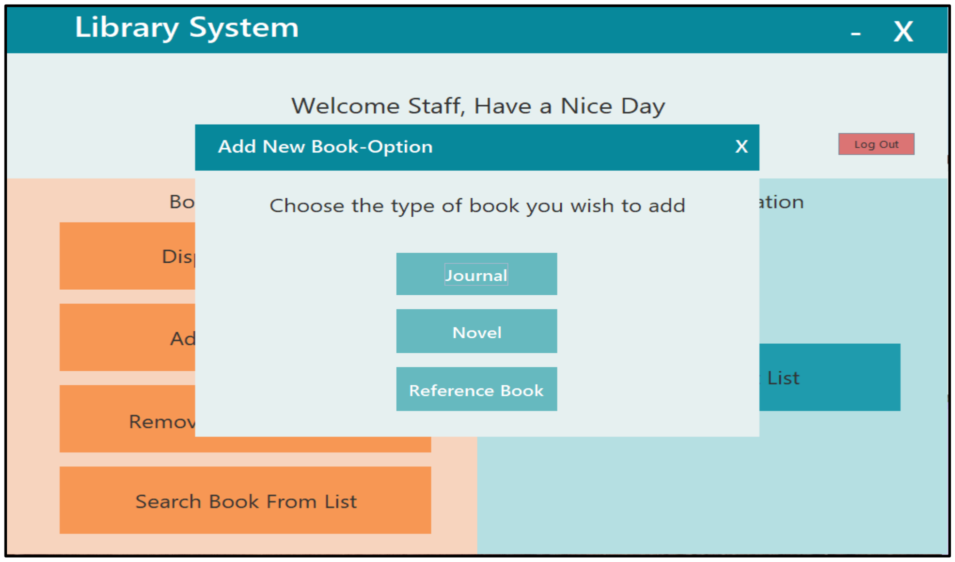
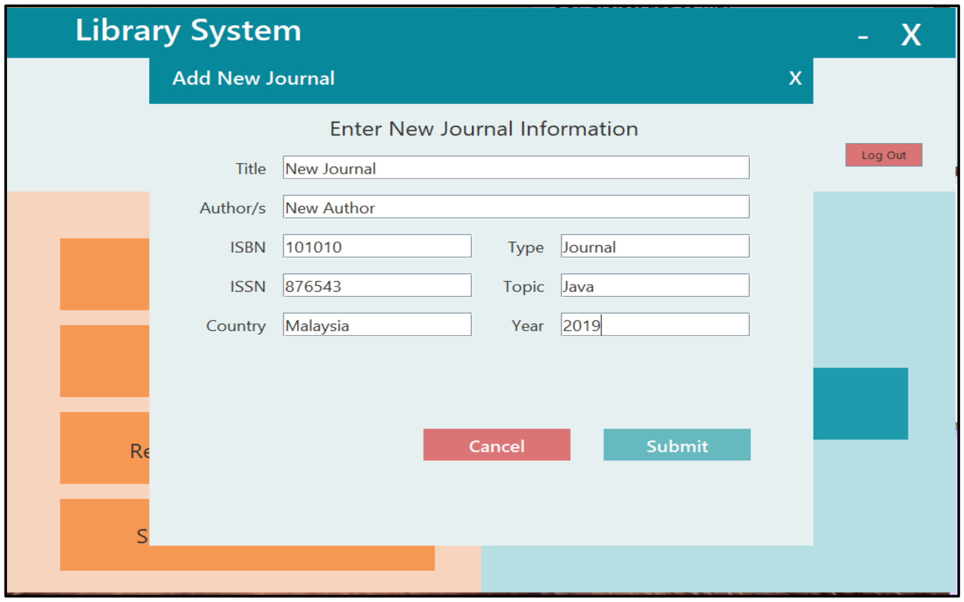


Diagram 39 show the type of new book for the staff to choose

6. In this case, we assume that the book that the staff wish to add is a journal. System displayed an interface that required staff to insert all the details about that book. For journal, novel and reference book, the interface is slightly different as their details is different.

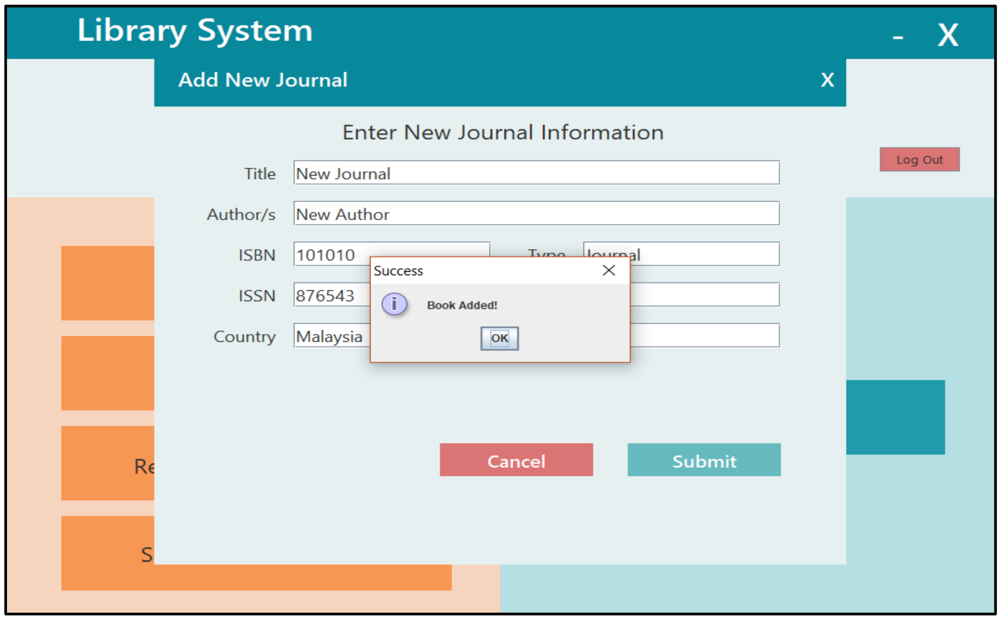


Enter New Journal Information

Title	New Journal		
Author/s	New Author		
ISBN	101010	Type	Journal
ISSN	876543	Topic	Java
Country	Malaysia	Year	2019

Diagram 40 show the interface for the journal.

7. Once the staff had completed the details, click submit and the system had successfully update the information, system will display an interface to inform the staff.



Enter New Journal Information

Title New Journal

Author/s New Author

ISBN 101010 Type Journal

ISSN 876543

Country Malaysia

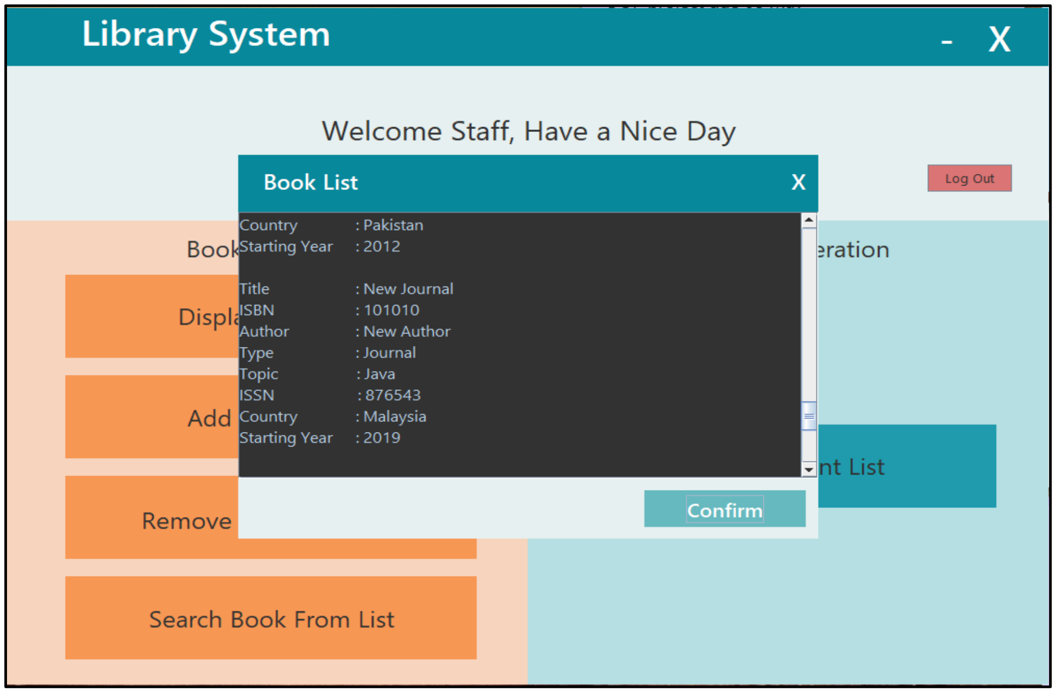
Success

Book Added!

OK

Diagram 41 show the interface that inform user the journal had successfully added.

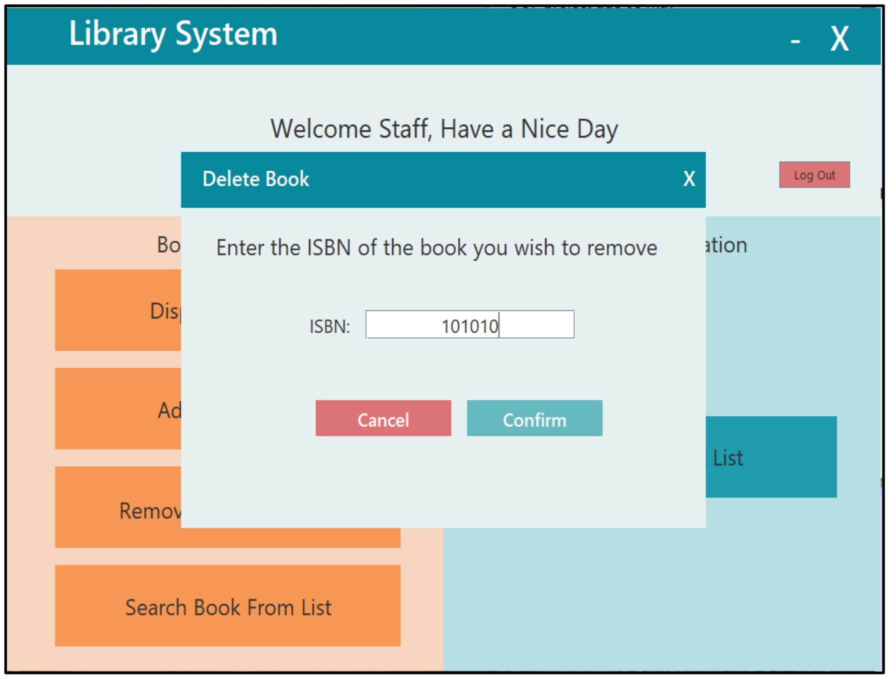
8. After that, the staff check the latest book list by selecting "**Display book list**". From the list, we noticed that the details of the new book had added into the list.



```
Country        : Pakistan
Starting Year  : 2012

Title          : New Journal
ISBN           : 101010
Author         : New Author
Type           : Journal
Topic          : Java
ISSN           : 876543
Country        : Malaysia
Starting Year  : 2019
```

Diagram 42 show that the new book had added to the list

9. Lastly, the staff can also remove the book from the list by select “**Remove book**”. Staff need to key in the ISBN of the book to enable system to trace, if the book found, system will remove it from the list.



Welcome Staff, Have a Nice Day

Enter the ISBN of the book you wish to remove

ISBN: 101010

Diagram 43 show that the ISBN had been key in by the staff in order to remove the book.