Library System

By Ng Jia Qi, Tan See Jou, Jovita Maulina Yusuf

Description

- The Library System aids librarian in managing the library and ease the students by providing self-service
- There is 2 perspective of this system, Student and Librarian(Staff)
- Student can borrow & return books and book study rooms
- Staff can manage books of library

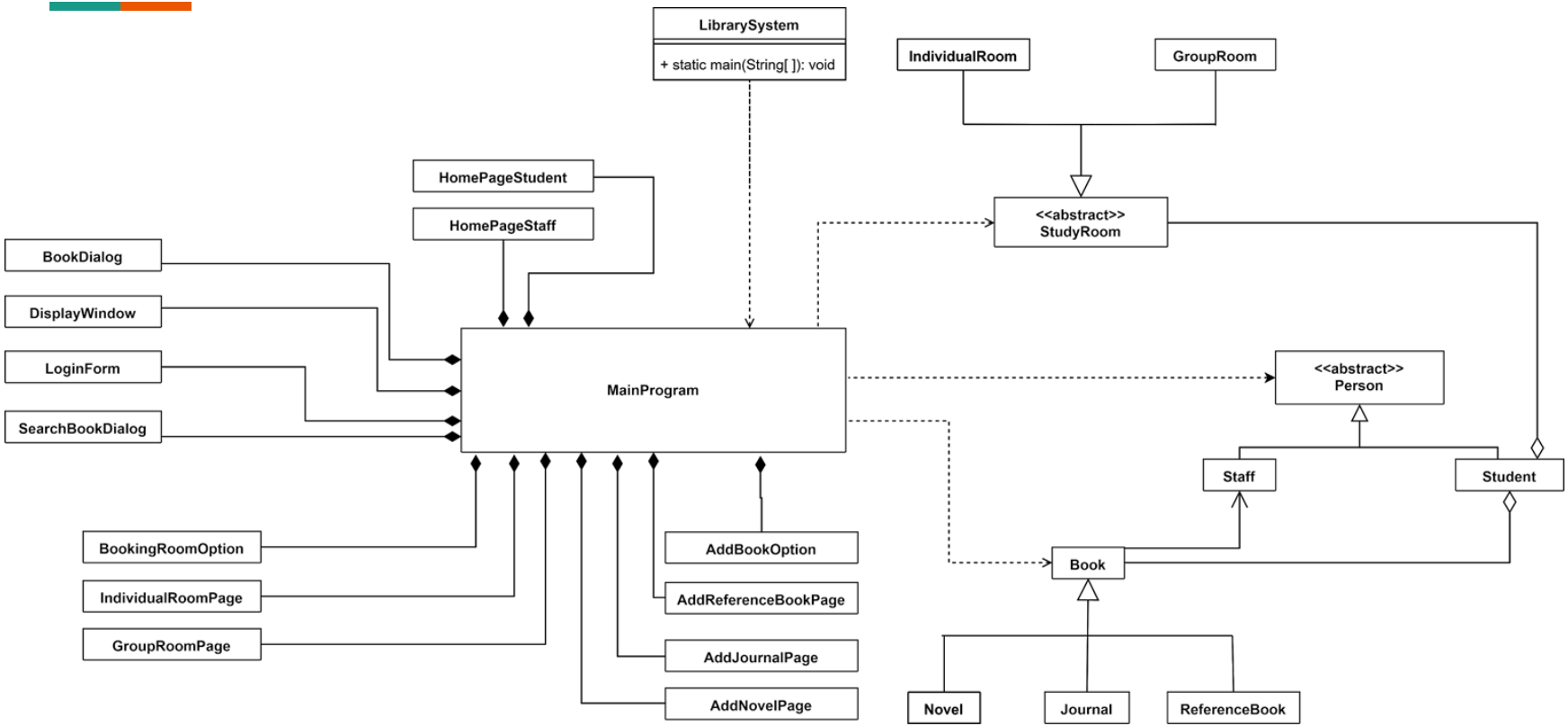
Student Functions

1. Borrow book
2. Return book
3. Check borrowed book
4. Search book
5. Booking study room
6. Check booking status
7. Check room list

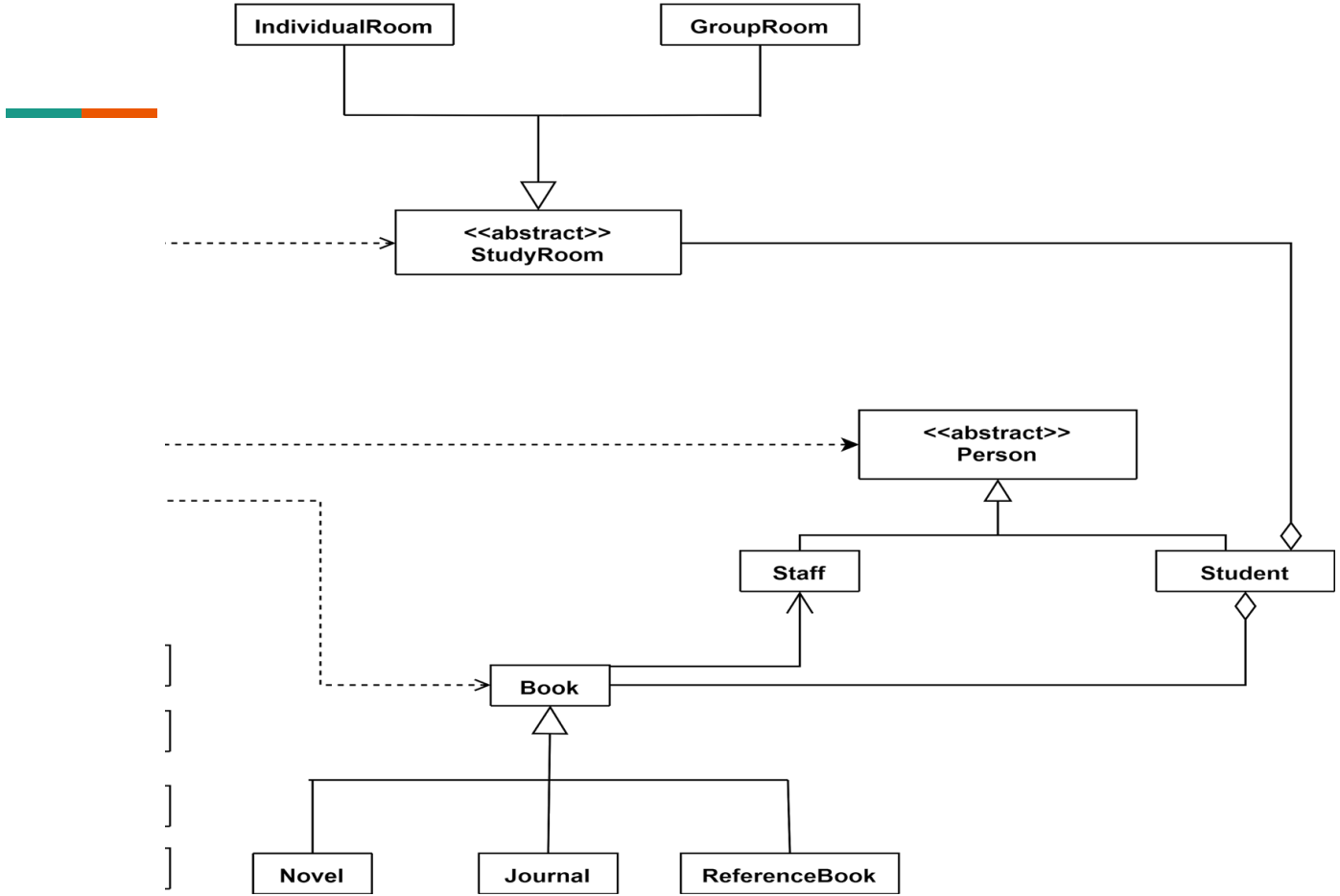
Staff Functions

1. Add book
2. Remove book
3. View book list
4. Search book
5. View student list

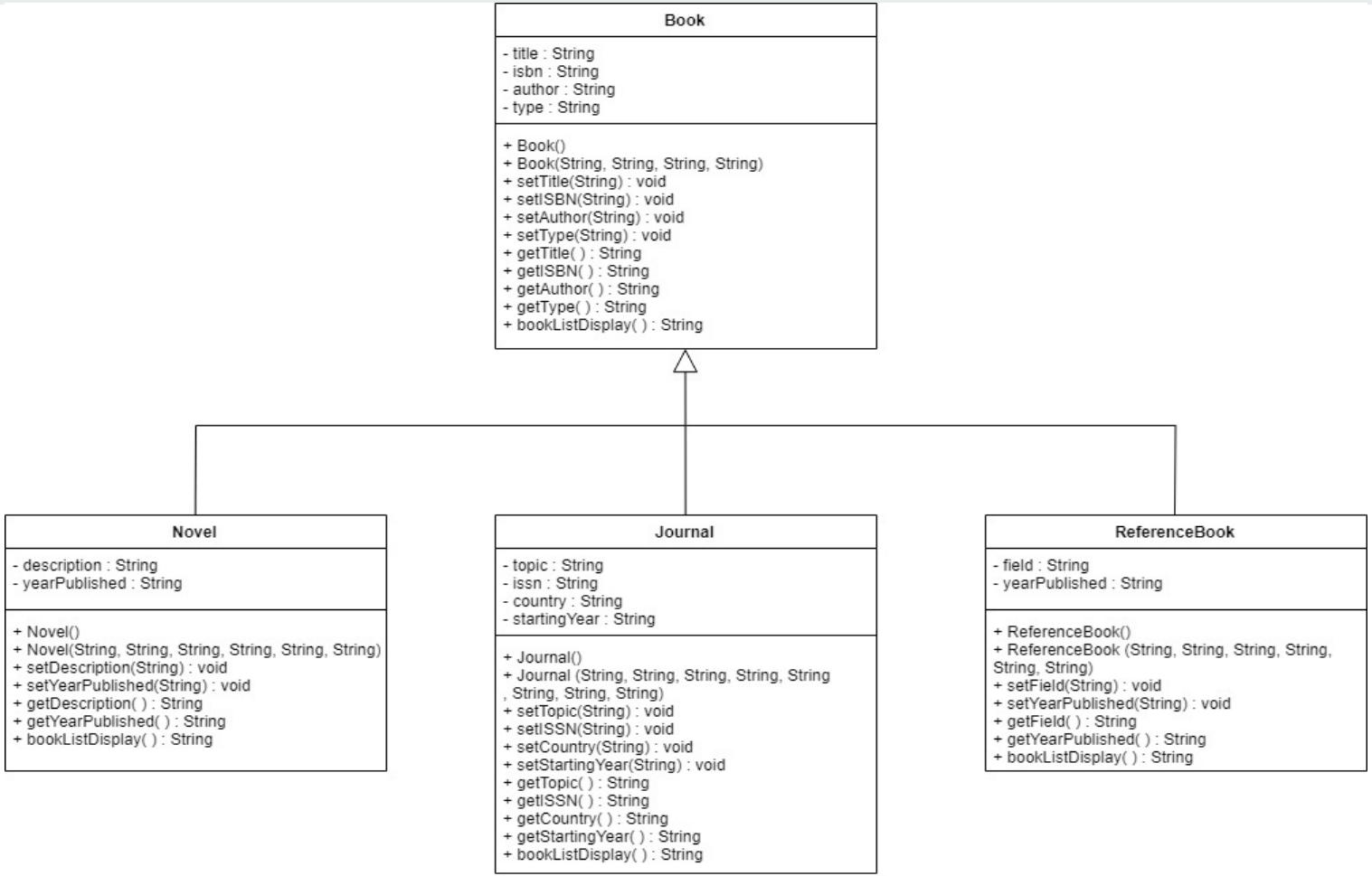
Class Diagram



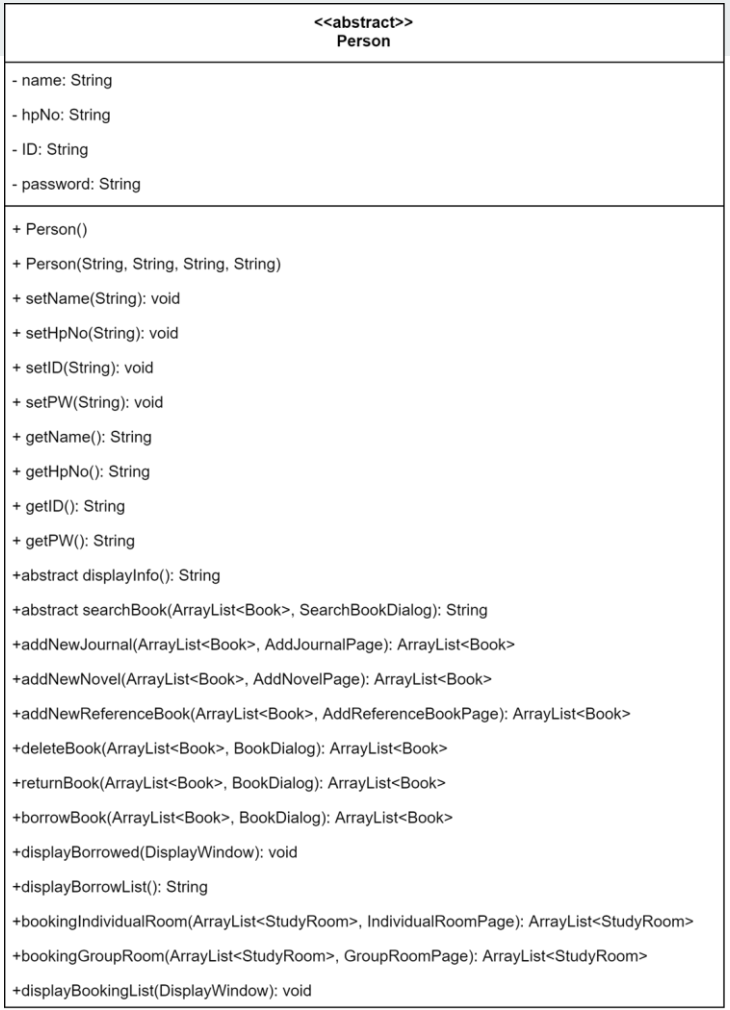
Let's Look Closer



Book Classes



Person Classes

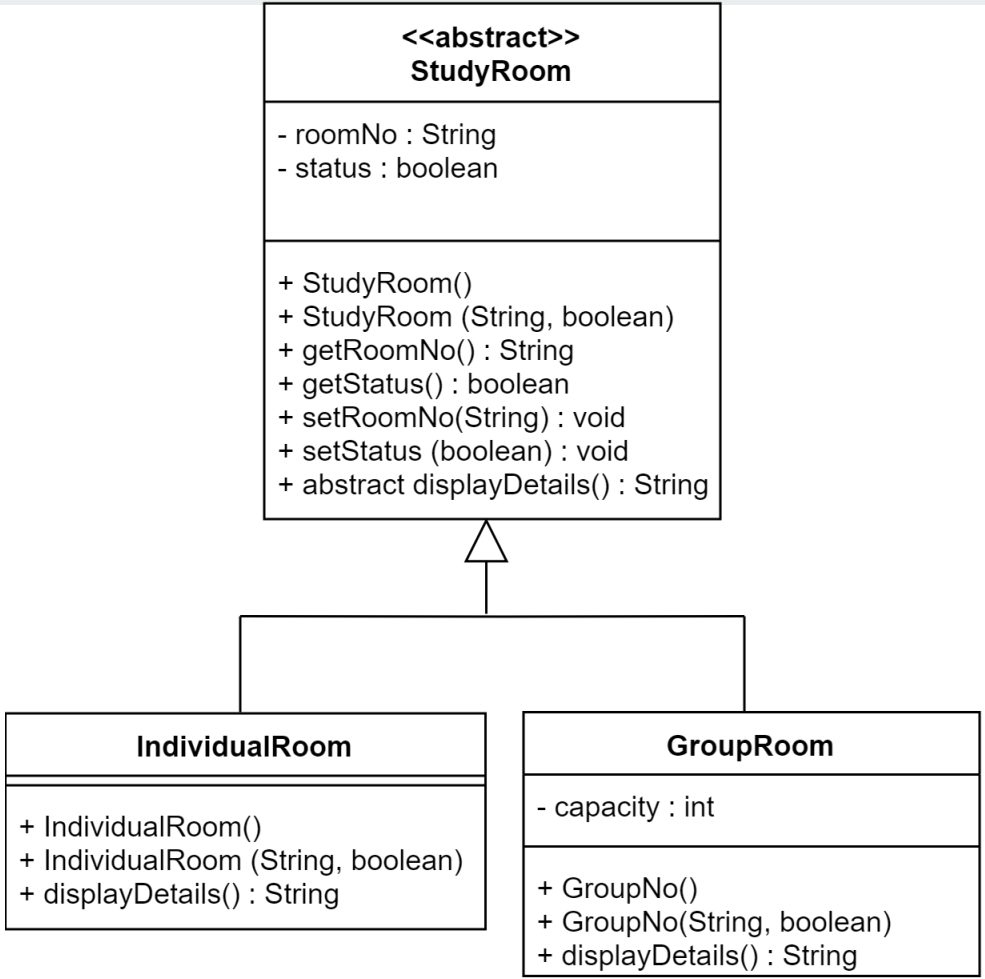


Continue..

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

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>

StudyRoom Classes



Concept Used

1. File Class
2. Encapsulation and Data Hiding
3. Class Relationship
4. Polymorphism
5. Exception Handling
6. ArrayList & Vector
7. Graphical User Interface (GUI)

File class

- Used to read the data or information of book from text file : **NovelList**, **JournalList**, **ReferenceBookList**.

```
130     public static ArrayList<Book> bookList()throws FileNotFoundException{
131         Scanner inputFile = new Scanner(new File("novelList.txt"));
132         ArrayList<Book> bList = new ArrayList<Book> ();
133         String title, isbn, author, type, desc, yearP, field, topic, issn, country, sYear;
134
135         //Read novel list
136         do{
137             title   = inputFile.nextLine();
138             isbn    = inputFile.nextLine();
139             author  = inputFile.nextLine();
140             type    = inputFile.nextLine();
141             desc    = inputFile.nextLine();
142             yearP   = inputFile.nextLine();
143             inputFile.nextLine();
144
145             Book b1 = new Novel(title, isbn, author, type, desc, yearP);
146             bList.add(b1);
147         }while(inputFile.hasNext());
148         inputFile.close();
149
```

Encapsulation and Data Hiding

- This concept had applied in all the classes so that a class can have total control of over its attribute and methods.

```
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 ;
    }
}
```

Class Relationship - Inheritance

- This concept we had applied in **Person** class and **Book** 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 ;
    }
}
```

Class Relationship - Aggregation

- This concept we had applied in between **Student** class and **Book** class.

```
6   class Student extends Person{
7       private Vector<Book> borrowList;
8       private int limit = 5;
9
10      public Student()
11      {
12
13      }
14      public Student(String name, String hpNo, String ID, String pw)
15      {
16          super(name, hpNo, ID, pw);
17          borrowList = new Vector<Book>();
18      }
19
```

Class Relationship - Aggregation

```
46      public void returnBook(ArrayList<Book> bookList){
47          System.out.print("Please enter the ISBN of the book you wish to return:");
48          Scanner input = new Scanner(System.in);
49          String isbn = input.nextLine();
50
51          int count=0;
52          for(int i=0; i< borrowList.size();i++){
53          if(borrowList.elementAt(i).getISBN().equals(isbn)){
54                      System.out.println("The book with ISBN " + isbn + " returned.");
55                      bookList.add(borrowList.get(i));
56                      borrowList.removeElementAt(i);
57              }
58              else{
59                  count++;
60              }
61          }
62          if(count>borrowList.size()){
63              System.out.println("The book with ISBN " + isbn + " not found. Please try again");
64          }
65      }
```

Polymorphism

- This concept we had applied in parent class and child class to reduce complexity of code viewing and also increase the usability of the code. .

```
5  abstract class Person{
6      ...
7      public abstract String displayInfo();
8      public abstract String searchBook(ArrayList<Book> bookList, SearchBookDialog searchBook);
9      public ArrayList<Book> addNewJournal(ArrayList<Book> bookList, AddJournalPage addJournalPage){return null;}
10     public ArrayList<Book> addNewNovel(ArrayList<Book> bookList, AddNovelPage addNovelPage){return null;}
11     public ArrayList<Book> addNewReferenceBook(ArrayList<Book> bookList, AddReferenceBookPage addReferenceBookPage)
12     {return null;}
13     public ArrayList<Book> deleteBook(ArrayList<Book> bookList, DeleteBookDialog deleteBook){return null;}
14     public void returnBook(ArrayList<Book> bookList){}
15     public void borrowBook(ArrayList<Book> bookList){}
16 }
```

Polymorphism

```
6  class Staff extends Person{
7      ...
8      //search book
9      public String searchBook(ArrayList<Book> bookList, SearchBookDialog searchBook){
10
11         String title = searchBook.getTitleInput().getText().toUpperCase();
12         for(int i=0; i< bookList.size();i++){
13             if(bookList.get(i).getTitle().toUpperCase().equals(title)){
14                 return "Book Found!\n"+ bookList.get(i).bookListDisplay();
15             }
16
17         }
18
19         JOptionPane.showMessageDialog(null,"Book Not Found!","Error",JOptionPane.INFORMATION_MESSAGE);
20         return null;
21     }
22 }
```

Exception Handling

- We had applied the exception handling concept in the **loginVerification().**

```
21      public boolean loginVerification(ArrayList<Person> userList)
22      {
23          try{
24              checkEmpty();
25              for(Person u: userList){
26                  if(idInput.getText().equals(u.getID()))
27                  {
28                      if(new String(pwInput.getPassword()).equals(u.getPW()))
29                          return true;
30                      else throw new Exception("Invalid Password...");
31                  }
32              }throw new Exception("User not exist...");
33
34          }
35          catch(Exception e)
36          {
37              JOptionPane.showMessageDialog(null, e.getMessage(),"Login Failed",JOptionPane.INFORMATION_MESSAGE);
38          }
39          return false;
40      }
```

Vector

- We had implement vector concept in **borrowList** in the **Student Class**.

```
31      public ArrayList<Book> borrowBook (ArrayList<Book> bookList, BookDialog studBorrowBook)
32      {
33          String isbn = studBorrowBook.getIsbnInput().getText();
34
35
36
37          for(int i=0; i<bookList.size(); i++)
38          {
39              if (bookList.get(i).getISBN().equals(isbn))
40              {
41                  if (checkLimit() == false)
42                  {
43                      JOptionPane.showMessageDialog(null,"The book with ISBN " + isbn + "borrowed."
44                                                    ,"Success",JOptionPane.INFORMATION_MESSAGE);
45                      borrowList.addElement(bookList.get(i));
46                      bookList.remove(i);
47                      return bookList;
48                  }
49                  else
50                      return bookList;
51              }
52          }
53          //if exceeds list size & didn't found that book
54          JOptionPane.showMessageDialog(null,"The book with ISBN " +isbn+ " not found. Please try again"
55                                        ,"Error", JOptionPane.INFORMATION_MESSAGE);
56          return bookList;
57
58      }
```

ArrayList

- We had implement **ArrayList** concept **BookList** due to its flexibility of size.

```
66    public static ArrayList<Book> bookList()throws FileNotFoundException{
67        Scanner inputFile = new Scanner(new File("novelList.txt"));
68        ArrayList<Book> bList = new ArrayList<Book> ();
69        String title, isbn, author, type, desc, yearP, field, topic, issn, country, sYear;
70
71        //Read novel list
72        do{
73            title   = inputFile.nextLine();
74            isbn    = inputFile.nextLine();
75            author  = inputFile.nextLine();
76            type    = inputFile.nextLine();
77            desc    = inputFile.nextLine();
78            yearP   = inputFile.nextLine();
79            inputFile.nextLine();
80
81            Book b1 = new Novel(title, isbn, author, type, desc, yearP);
82            bList.add(b1);
83        }while(inputFile.hasNext());
84        inputFile.close();
85
86        ...
```

GUI

- Java libraries that has been implement included **javax.swing** and **java.awt.event**

```
136     //booking individual studyroom
137     public ArrayList<StudyRoom> bookingIndividualRoom(ArrayList<StudyRoom> roomList,IndividualRoomPage bookRoom)
138     {
139         String roomNo = bookRoom.getRoomChoice().getSelectedItem().toString();
140         if(roomNo.equals("--")){
141             JOptionPane.showMessageDialog(null,"Please choose a room..."
142                                           ,"Error", JOptionPane.INFORMATION_MESSAGE);
143             return roomList;
144         }
145         for(int i=0;i<roomList.size();i++){
146             if(roomList.get(i).getRoomNo().equals(roomNo)&&roomList.get(i).getStatus().equals("Available")){
147                 JOptionPane.showMessageDialog(null,"The room with room no. " + roomNo + " booked."
148                                               ,"Success",JOptionPane.INFORMATION_MESSAGE);
149                 studyRoom = roomList.get(i);
150                 roomList.get(i).setStatus(false);
151                 return roomList;
152             }
153             if(roomList.get(i).getRoomNo().equals(roomNo)&&roomList.get(i).getStatus().equals("Not Available")){
154                 JOptionPane.showMessageDialog(null,"Please choose an available room..."
155                                               ,"Error", JOptionPane.INFORMATION_MESSAGE);
156             }
157 
158         }
159         return roomList;
160     }
```

DEMO...