TITLE: Data Structure & Algorithms

STUDENT NAME: Alisson Alves De Moura.

STUDENT NUMBER: 2019142

LECTURER: Ms. Aldana Louzan.

Link to GitHub: https://github.com/AlissonDMoura/LibraryTerminal

Content

Much Obliged !

Objective:

This assessment objective is to create an Java Machine that utilizes concepts and structures such as queues, searches and sorts that utilizes a new method created by the Student. Utilizing knowledge of Data structures to overcome the challenge

The challenge proposed was to create an artefact for a library that shall use it as its new Database to write and read from csv files and proceed with borrows and control Readers and book in a friendly interface that runs in simple computers.

1. Program Structure

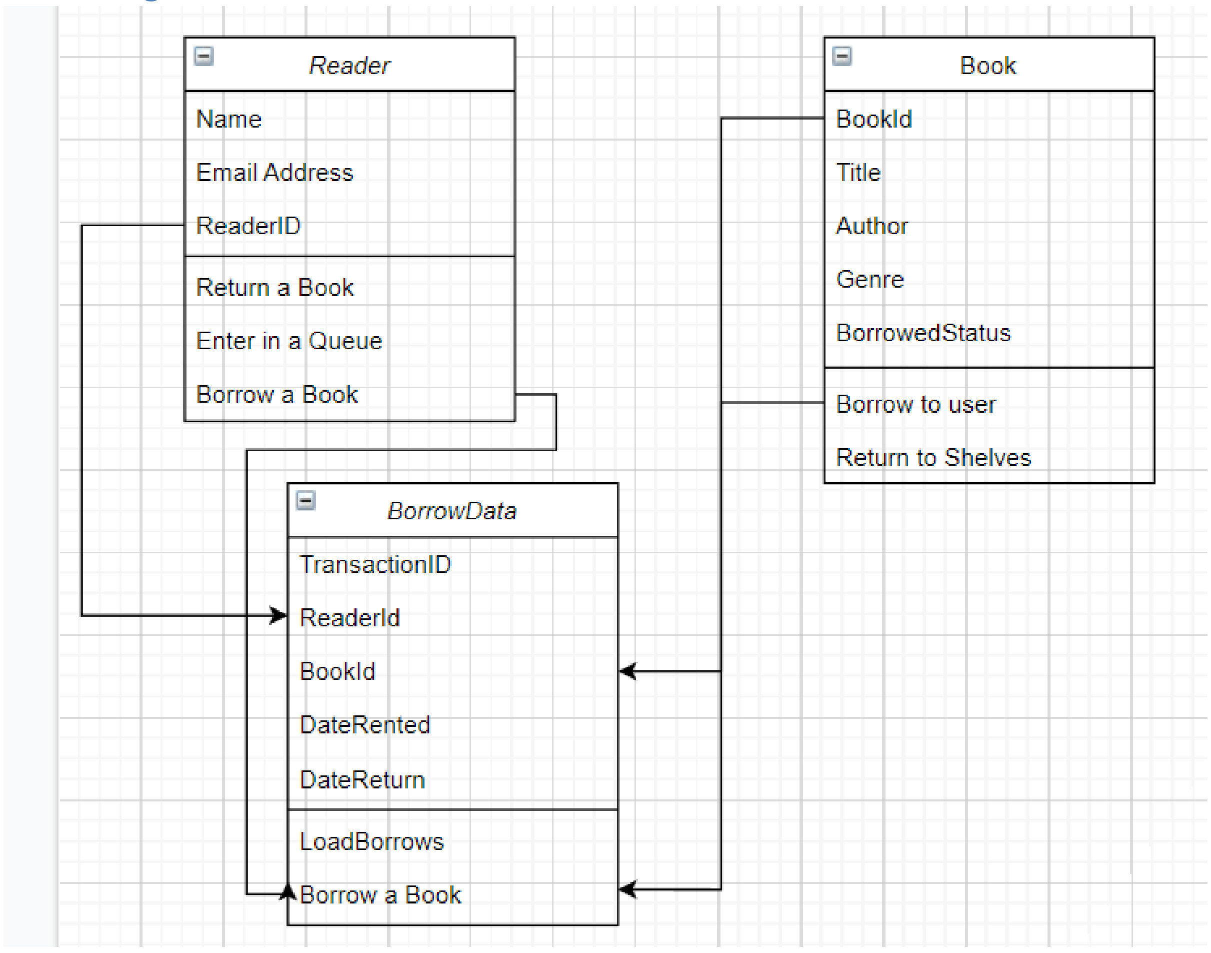


Figure 1 – Main Classes Objects

The Structure of the program utilizes of a total of eight classes, being three Classes there are capable of creating new main objects, A class to create Nodes for a queue, A class that operate queues and three other classes for controls and interactivity with data and the user.

The main challenge was to develop new methods to operate the task and the challenge of creating and using a system to store offline information, making it tricky to reduce the usage of processing power; writing data from file to memory, updating it and rewriting into storage safely.

Another main challenge was to validate all information from user, and reduce interactivity to user to the least necessary in order to avoid modifications to the data entered in the system.

Thought the code, It will possible to visualize the number of Catch exceptions is great due to the frequency the data in the csv file or in the memory changes and have to be properly processed to keep the program running.

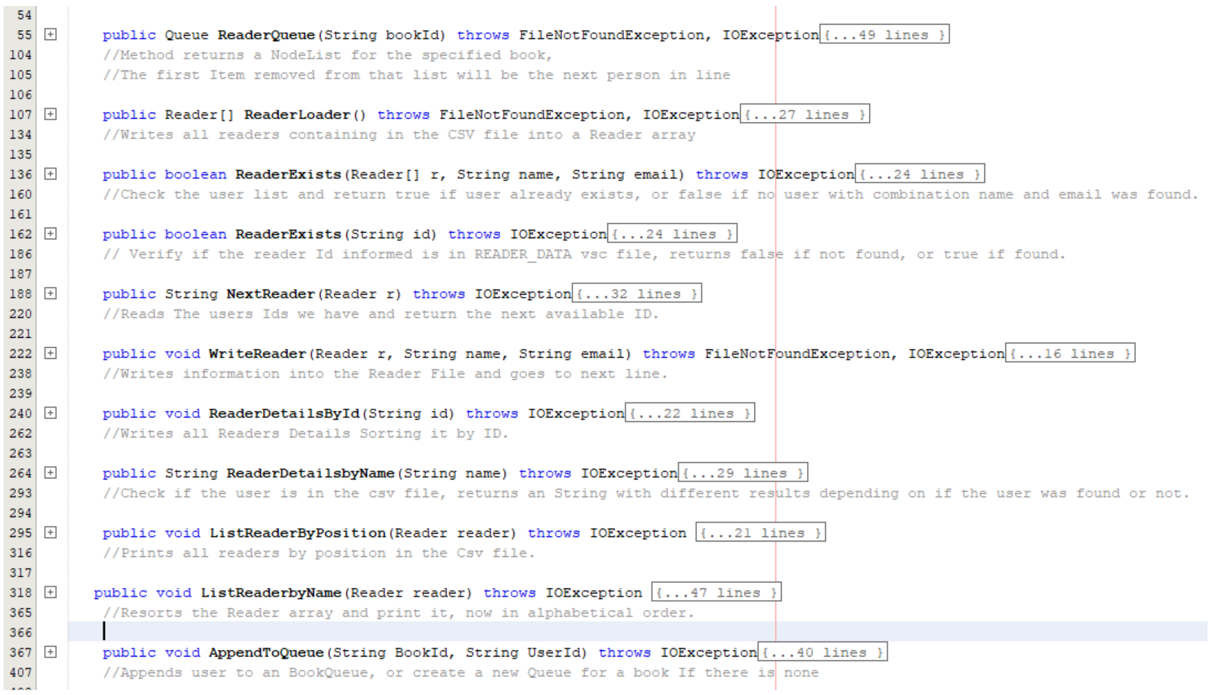
The only exit to the program was thought to be also in the menu, every other interaction was created to be validated to keep the program running.

2. Commented Code.

The code was commented properly for each interaction with data and user.

This is of vital importance and very helpful to visualize the plan of the student whilst programming and keep focus in the task at hand.
Even thought this is an aspect due for grading, It was also vital to make the program be done by the submission date.

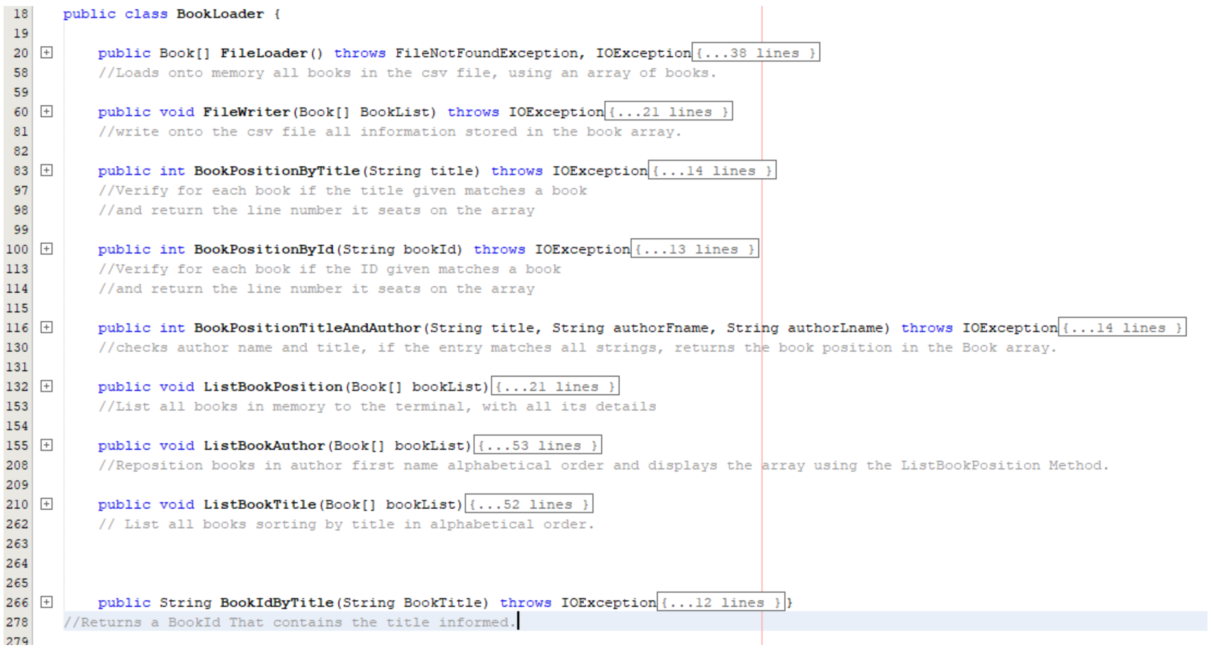
By Commenting the code, that became substantially big due to so many different methods that complement each other, It was easy to find mistakes and correct them properly whenever necessary.



```
    public Queue ReaderQueue(String bookId) throws FileNotFoundException, IOException{...49 lines }
    //Method returns a NodeList for the specified book,
    //The first Item removed from that list will be the next person in line

    public Reader[] ReaderLoader() throws FileNotFoundException, IOException{...27 lines }
    //Writes all readers containing in the CSV file into a Reader array

    public boolean ReaderExists(Reader[] r, String name, String email) throws IOException{...24 lines }
    //Check the user list and return true if user already exists, or false if no user with combination name and email was found.

    public boolean ReaderExists(String id) throws IOException{...24 lines }
    // Verify if the reader Id informed is in READER_DATA vsc file, returns false if not found, or true if found.

    public String NextReader(Reader r) throws IOException{...32 lines }
    //Reads The users Ids we have and return the next available ID.

    public void WriteReader(Reader r, String name, String email) throws FileNotFoundException, IOException{...16 lines }
    //Writes information into the Reader File and goes to next line.

    public void ReaderDetailsById(String id) throws IOException{...22 lines }
    //Writes all Readers Details Sorting it by ID.

    public String ReaderDetailsbyName(String name) throws IOException{...29 lines }
    //Check if the user is in the csv file, returns an String with different results depending on if the user was found or not.

    public void ListReaderByPosition(Reader reader) throws IOException {...21 lines }
    //Prints all readers by position in the Csv file.

   public void ListReaderbyName(Reader reader) throws IOException {...47 lines }
    //Resorts the Reader array and print it, now in alphabetical order.

    public void AppendToQueue(String BookId, String UserId) throws IOException{...40 lines }
    //Appends user to an BookQueue, or create a new Queue for a book If there is none
```

Figure 2 – Auxiliary Methods in Reader Class

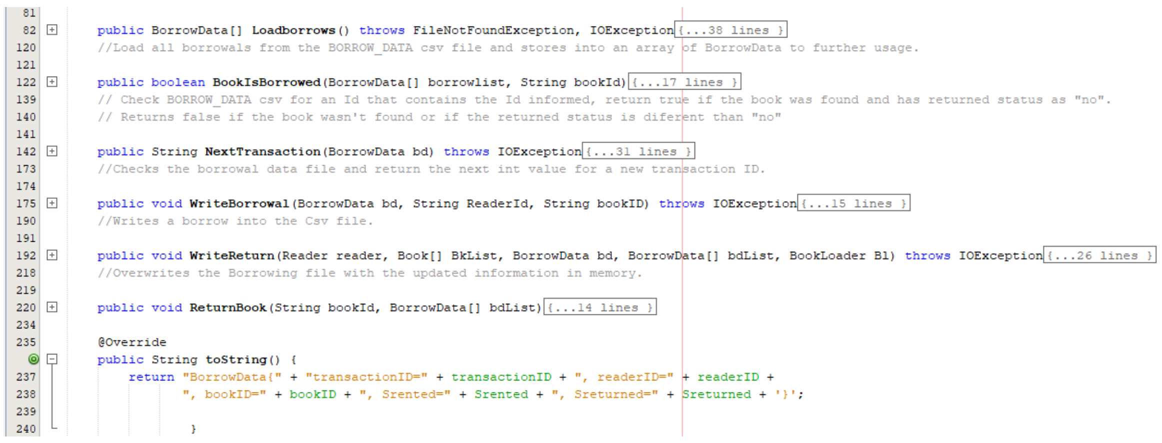
The Reader Class Has methods to work with readers in Different situations, It contains Queues of readers waiting to Borrow a book, Load data about readers from csv file, Validates if a reader exists, Write back to the csv File, List and Search Readers.



```java
public class BookLoader {

    public Book[] FileLoader() throws FileNotFoundException, IOException{...38 lines }
    //Loads onto memory all books in the csv file, using an array of books.

    public void FileWriter(Book[] BookList) throws IOException{...21 lines }
    //write onto the csv file all information stored in the book array.

    public int BookPositionByTitle(String title) throws IOException{...14 lines }
    //Verify for each book if the title given matches a book
    //and return the line number it seats on the array

    public int BookPositionById(String bookId) throws IOException{...13 lines }
    //Verify for each book if the ID given matches a book
    //and return the line number it seats on the array

    public int BookPositionTitleAndAuthor(String title, String authorFname, String authorLname) throws IOException{...14 lines }
    //checks author name and title, if the entry matches all strings, returns the book position in the Book array.

    public void ListBookPosition(Book[] bookList){...21 lines }
    //List all books in memory to the terminal, with all its details

    public void ListBookAuthor(Book[] bookList){...53 lines }
    //Reposition books in author first name alphabetical order and displays the array using the ListBookPosition Method.

    public void ListBookTitle(Book[] bookList){...52 lines }
    // List all books sorting by title in alphabetical order.



    public String BookIdByTitle(String BookTitle) throws IOException{...12 lines }}
//Returns a BookId That contains the title informed.
```

Figure 3 – BookLoader Class Methods.

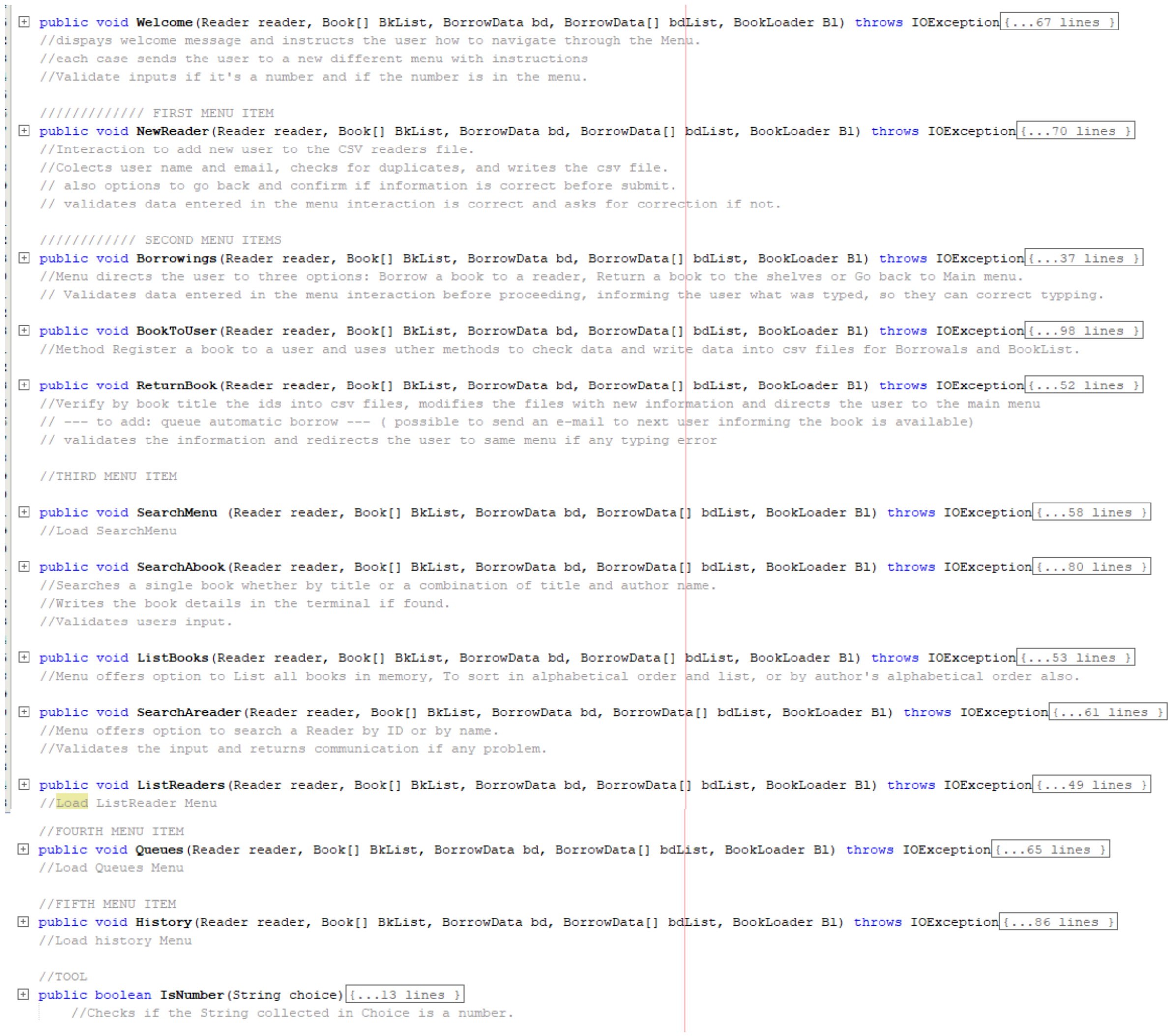
The BookLoader Class Work with books; Loads an Book Array containing all books in the Library, has Methods to interact with them for Search, validations and lists, read and writes from and to the csv File.



```java

    public BorrowData[] Loadborrows() throws FileNotFoundException, IOException{...38 lines }
    //Load all borrowals from the BORROW_DATA csv file and stores into an array of BorrowData to further usage.

    public boolean BookIsBorrowed(BorrowData[] borrowlist, String bookId){...17 lines }
    // Check BORROW_DATA csv for an Id that contains the Id informed, return true if the book was found and has returned status as "no".
    // Returns false if the book wasn't found or if the returned status is diferent than "no"

    public String NextTransaction(BorrowData bd) throws IOException{...31 lines }
    //Checks the borrowal data file and return the next int value for a new transaction ID.

    public void WriteBorrowal(BorrowData bd, String ReaderId, String bookID) throws IOException{...15 lines }
    //Writes a borrow into the Csv file.

    public void WriteReturn(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...26 lines }
    //Overwrites the Borrowing file with the updated information in memory.

    public void ReturnBook(String bookId, BorrowData[] bdList){...14 lines }

    @Override
    public String toString() {
        return "BorrowData{" + "transactionID=" + transactionID + ", readerID=" + readerID +
                ", bookID=" + bookID + ", Srented=" + Srented + ", Sreturned=" + Sreturned + '}';

                 }
```

Figure 4 – BorrowData Class Methods.

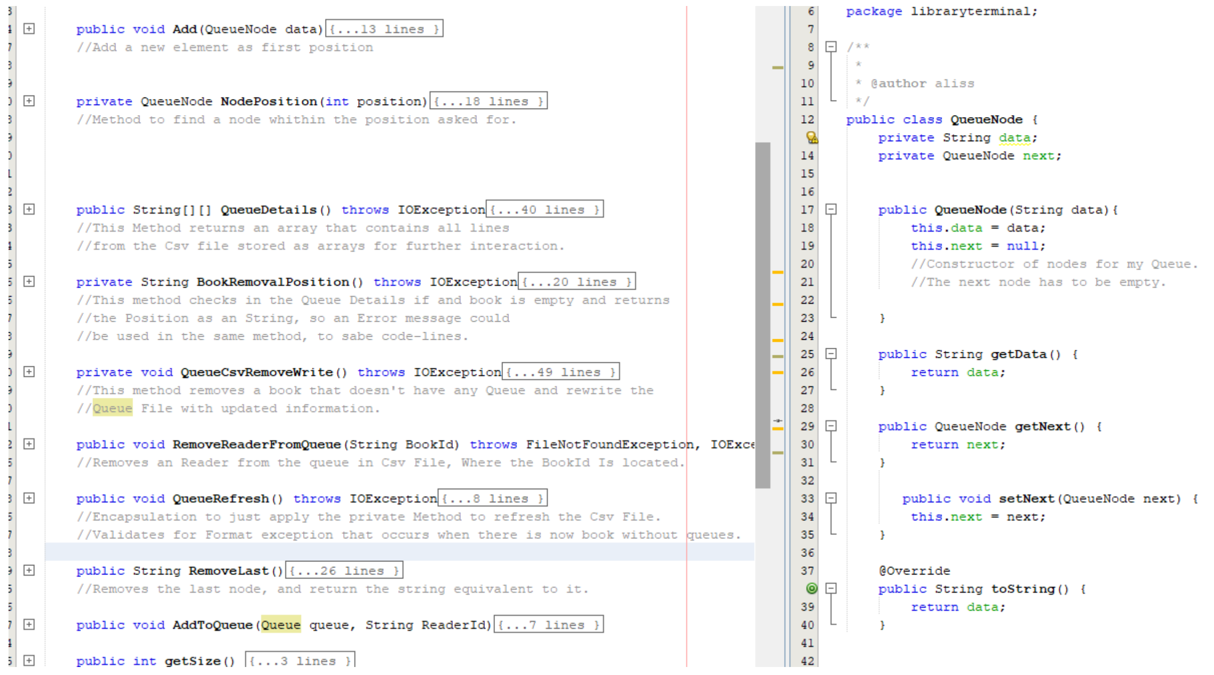
The BorrowData Class work with transactions, It Loads the borrow history in an array, Validates the information, Append new borrow, read and write from and to the Csv Files.



```
public void Welcome(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...67 lines }
    //dispays welcome message and instructs the user how to navigate through the Menu.
    //each case sends the user to a new different menu with instructions
    //Validate inputs if it's a number and if the number is in the menu.

    //////////// FIRST MENU ITEM
public void NewReader(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...70 lines }
    //Interaction to add new user to the CSV readers file.
    //Colects user name and email, checks for duplicates, and writes the csv file.
    // also options to go back and confirm if information is correct before submit.
    // validates data entered in the menu interaction is correct and asks for correction if not.

    //////////// SECOND MENU ITEMS
public void Borrowings(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...37 lines }
    //Menu directs the user to three options: Borrow a book to a reader, Return a book to the shelves or Go back to Main menu.
    // Validates data entered in the menu interaction before proceeding, informing the user what was typed, so they can correct typping.

public void BookToUser(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...98 lines }
    //Method Register a book to a user and uses uther methods to check data and write data into csv files for Borrowals and BookList.

public void ReturnBook(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...52 lines }
    //Verify by book title the ids into csv files, modifies the files with new information and directs the user to the main menu
    // --- to add: queue automatic borrow --- ( possible to send an e-mail to next user informing the book is available)
    // validates the information and redirects the user to same menu if any typing error

    //THIRD MENU ITEM

public void SearchMenu (Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...58 lines }
    //Load SearchMenu

public void SearchAbook(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...80 lines }
    //Searches a single book whether by title or a combination of title and author name.
    //Writes the book details in the terminal if found.
    //Validates users input.

public void ListBooks(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...53 lines }
    //Menu offers option to List all books in memory, To sort in alphabetical order and list, or by author's alphabetical order also.

public void SearchAreader(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...61 lines }
    //Menu offers option to search a Reader by ID or by name.
    //Validates the input and returns communication if any problem.

public void ListReaders(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...49 lines }
    //Load ListReader Menu

    //FOURTH MENU ITEM
public void Queues(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...65 lines }
    //Load Queues Menu

    //FIFTH MENU ITEM
public void History(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{...86 lines }
    //Load history Menu

    //TOOL
public boolean IsNumber(String choice){...13 lines }
        //Checks if the String collected in Choice is a number.
```

Figure 5 – Class Interactor Methods

The Class Interactor was created entirely to create user interactivity with the program.
Every method carries the same information stored in memory, and interact with each other, as the parameters the methods utilize was standardized in this class.
In an elegant way, The navigation through the menu was easy linked whilst the back end works the methods.

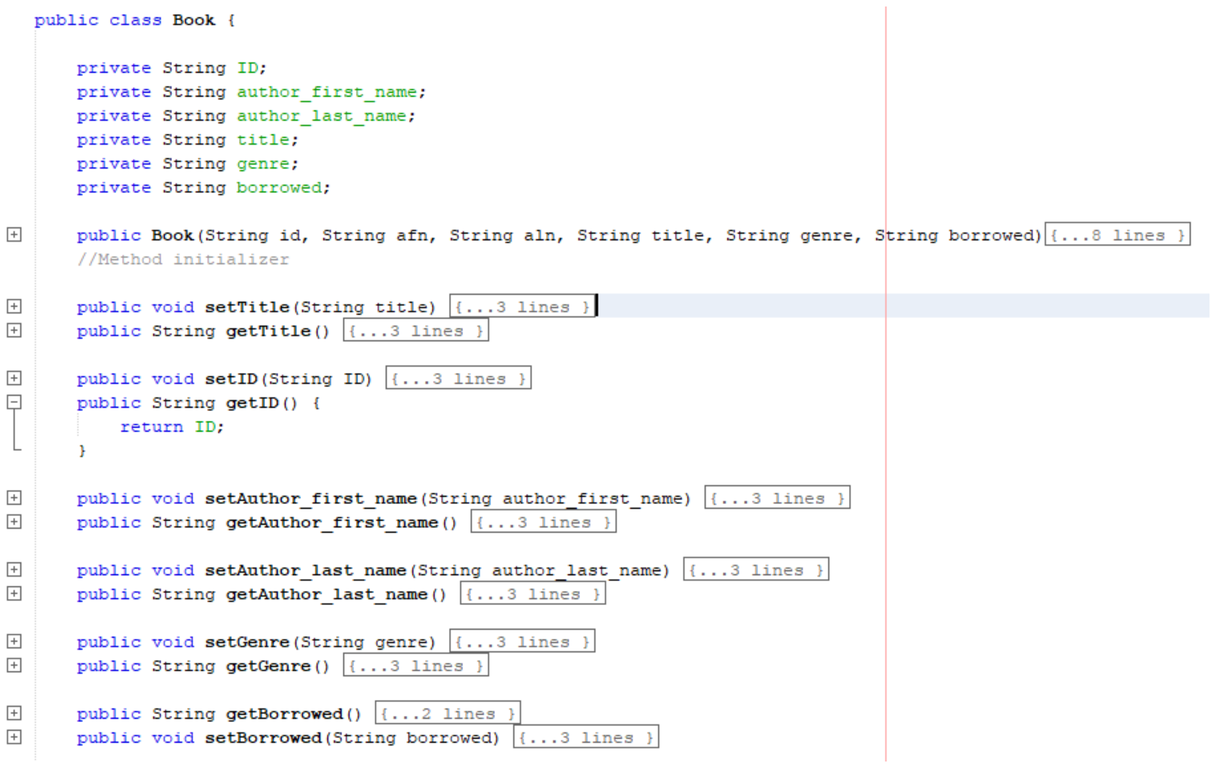


```java
    public void Add(QueueNode data){...13 lines }
    //Add a new element as first position


    private QueueNode NodePosition(int position){...18 lines }
    //Method to find a node whithin the position asked for.




    public String[][] QueueDetails() throws IOException{...40 lines }
    //This Method returns an array that contains all lines
    //from the Csv file stored as arrays for further interaction.

    private String BookRemovalPosition() throws IOException{...20 lines }
    //This method checks in the Queue Details if and book is empty and returns
    //the Position as an String, so an Error message could
    //be used in the same method, to sabe code-lines.

    private void QueueCsvRemoveWrite() throws IOException{...49 lines }
    //This method removes a book that doesn't have any Queue and rewrite the
    //Queue File with updated information.

    public void RemoveReaderFromQueue(String BookId) throws FileNotFoundException, IOExce
    //Removes an Reader from the queue in Csv File, Where the BookId Is located.

    public void QueueRefresh() throws IOException{...8 lines }
    //Encapsulation to just apply the private Method to refresh the Csv File.
    //Validates for Format exception that occurs when there is now book without queues.

    public String RemoveLast(){...26 lines }
    //Removes the last node, and return the string equivalent to it.

    public void AddToQueue(Queue queue, String ReaderId){...7 lines }

    public int getSize() {...3 lines }
```

```java
package libraryterminal;

/**
 *
 * @author aliss
 */
public class QueueNode {
    private String data;
    private QueueNode next;


    public QueueNode(String data){
        this.data = data;
        this.next = null;
        //Constructor of nodes for my Queue.
        //The next node has to be empty.

    }

    public String getData() {
        return data;
    }

    public QueueNode getNext() {
        return next;
    }

       public void setNext(QueueNode next) {
        this.next = next;
    }

    @Override
    public String toString() {
        return data;
    }


```

Figure 6 – Queue and QueueNode Classes

The class Queue and Queuenode Work to Create an queue, read and write from the csv File and interact with the data with several different validations.
This does a simple job: Make sure a Queue is created if necessary, Delete a queue if necessary, Edit an queue if necessary.



```java
public class Book {

    private String ID;
    private String author_first_name;
    private String author_last_name;
    private String title;
    private String genre;
    private String borrowed;

    public Book(String id, String afn, String aln, String title, String genre, String borrowed){...8 lines }
    //Method initializer

    public void setTitle(String title) {...3 lines }
    public String getTitle() {...3 lines }

    public void setID(String ID) {...3 lines }
    public String getID() {
        return ID;
    }

    public void setAuthor_first_name(String author_first_name) {...3 lines }
    public String getAuthor_first_name() {...3 lines }

    public void setAuthor_last_name(String author_last_name) {...3 lines }
    public String getAuthor_last_name() {...3 lines }

    public void setGenre(String genre) {...3 lines }
    public String getGenre() {...3 lines }

    public String getBorrowed() {...2 lines }
    public void setBorrowed(String borrowed) {...3 lines }
```

Figure 7 – Book Class.

The Less Commented Class is Book, That only stores data, has setters and getters and an simple method to self-initialize.

3. The Code.

```
Main:public static void main(String[] args) throws IOException { // TODO code application logic
here Interactor interactor = new Interactor(); Reader reader = new Reader(); BorrowData
bd = new BorrowData(); BorrowData[] bdList = bd.Loadborrows(); BookLoader bl = new
BookLoader(); Book[] BkList = bl.FileLoader(); interactor.Welcome(reader, BkList, bd,
bdList, bl);}public class Book { private String ID; private String author_first_name; private String
author_last_name; private String title; private String genre; private String borrowed; public
Book(String id, String afn, String aln, String title, String genre, String borrowed){ this.ID = id;
this.author_first_name = afn; this.author_last_name = aln; this.title = title; this.genre =
genre; this.borrowed = borrowed; } //Method initializer public void setTitle(String title) {
this.title = title; } public String getTitle() { return title; } public void setID(String ID) {
this.ID = ID; } public String getID() { return ID; } public void setAuthor_first_name(String
author_first_name) { this.author_first_name = author_first_name; } public String
getAuthor_first_name() { return author_first_name; } public void setAuthor_last_name(String
author_last_name) { this.author_last_name = author_last_name; } public String
getAuthor_last_name() { return author_last_name; } public void setGenre(String genre) {
this.genre = genre; } public String getGenre() { return genre; } public String getBorrowed()
{ return borrowed;} public void setBorrowed(String borrowed) { this.borrowed = borrowed;
} @Override public String toString() { return "Book{" + "ID=" + ID + ", author_first_name=" +
author_first_name + ", author_last_name=" + author_last_name + ", title=" + title + ", genre=" +
genre + ", borrowed=" + borrowed + '}'; } public class BookLoader { public Book[] FileLoader()
throws FileNotFoundException, IOException{ FileReader file = new
FileReader("BOOK_DATA.csv"); BufferedReader br = new BufferedReader(file); String read =
null; String[] data; Book[] BookArray = new Book[501]; int i=0; while((read =
br.readLine())!=null) {data = read.split(",,"); // //------------ String id = data[0];
String author_first_name = data[1]; String author_last_name = data[2]; String title = data[3];
String genre = data[4]; String borrowed = data[5]; Book book = new Book(id,
author_first_name,author_last_name,title, genre, borrowed ); BookArray[i] = book; i++; }
return BookArray;} //Loads onto memory all books in the csv file, using an array of books. public
void FileWriter(Book[] BookList) throws IOException{ int i = 0; FileWriter fw = new
FileWriter("TEST_DUMMY.csv", false); while(i < BookList.length){
fw.append(BookList[i].getID() + ",,"); fw.append(BookList[i].getAuthor_first_name() + ",,");
fw.append(BookList[i].getAuthor_last_name() + ",,"); fw.append(BookList[i].getTitle() + ",,");
fw.append(BookList[i].getGenre() + ",,"); fw.append(BookList[i].getBorrowed() + "\n");
i++; } fw.flush(); fw.close(); } //write onto the csv file all information stored in the
book array. public int BookPositionByTitle(String title) throws IOException{ int i = 0; Book[]
booklist = FileLoader(); while(i < booklist.length){
if(booklist[i].getTitle().equalsIgnoreCase(title)){ return i;} i++; }
return 0; } //Verify for each book if the title given matches a book //and return the line number
it seats on the array public int BookPositionById(String bookId) throws IOException{ int i = 0;
Book[] booklist = FileLoader(); while(i < booklist.length){
if(booklist[i].getID().equalsIgnoreCase(bookId)){ return i;} i++; } return
0; } //Verify for each book if the ID given matches a book //and return the line number it seats
on the array public int BookPositionTitleAndAuthor(String title, String authorFname, String
authorLname) throws IOException{        int i = 0;        Book[] booklist = FileLoader();        while(i <
booklist.length){                              if(booklist[i].getTitle().equalsIgnoreCase(title)    &&
booklist[i].getAuthor_first_name().equals(authorFname)                                            &&
booklist[i].getAuthor_last_name().equals(authorLname)){            return i;}        i++;                }
return 0;    }    //checks author name and title, if the entry matches all strings, returns the book
position    in    the    Book    array.            public    void    ListBookPosition(Book[]    bookList){
System.out.println("LISTING  ALL  BOOKS  IN  MEMORY: \n");            int  i  =  0;            while(i  <
bookList.length){                         System.out.println("Book   Id:   "   +   bookList[i].getID()+"\n"
+"Author First Name: "+ bookList[i].getAuthor_first_name() + "\n"                        +"Author Last
Name: "+ bookList[i].getAuthor_last_name() + "\n"                 +"Title: " + bookList[i].getTitle() +
"\n"                   +"Genre: " + bookList[i].getGenre() + "\n"                     +"Book is Borrowed
status: " +bookList[i].getBorrowed()+"\n");        i++;      }System.out.println("----** End of List **----
");}       //List  all  books  in  memory  to  the  terminal,  with  all  its  details       public  void
ListBookAuthor(Book[]  bookList){          String  tempId;          String  tempFirstName;          String
tempLastName;      String tempTitle;      String tempGenre;      String tempBorrowed;      for (int i
= 0; i < bookList.length; i++) {          for (int j = i + 1; j < bookList.length; j++) {            // to compare
one      string      with      other      strings                                                           if
(bookList[i].getAuthor_first_name().compareTo(bookList[j].getAuthor_first_name())      >      0)      {
// swapping                         tempId  =  bookList[i].getID();                          tempFirstName  =
bookList[i].getAuthor_first_name();               tempLastName = bookList[i].getAuthor_last_name();
tempTitle  =  bookList[i].getTitle();                               tempGenre  =  bookList[i].getGenre();
tempBorrowed = bookList[i].getBorrowed();                //Saves all information stored in Position i
into  an  temporary  array.                                      bookList[i].setID(bookList[j].getID());
bookList[i].setAuthor_first_name(bookList[j].getAuthor_first_name());
bookList[i].setAuthor_last_name(bookList[j].getAuthor_last_name());
bookList[i].setTitle(bookList[j].getTitle());                bookList[i].setGenre(bookList[j].getGenre());
bookList[i].setBorrowed(bookList[j].getBorrowed());             //Transfers all Strings from position j
to   Position   i                                                            bookList[j].setID(tempId);
bookList[j].setAuthor_first_name(tempFirstName);
bookList[j].setAuthor_last_name(tempLastName);                          bookList[j].setTitle(tempTitle);
bookList[j].setGenre(tempGenre);                              bookList[j].setBorrowed(tempBorrowed);
//Finishes the swap.            }          }       }           // print output array       System.out.println(
"-- ** LIST OF BOOKS IN AUTHORS ALPHABETICAL ORDER ** -- ");          ListBookPosition(bookList);}
//Reposition  books  in  author  first  name  alphabetical  order  and  displays  the  array  using  the
ListBookPosition Method.    public void ListBookTitle(Book[] bookList){      String tempId;      String
tempFirstName;      String tempLastName;      String tempTitle;      String tempGenre;      String
tempBorrowed;for (int i = 0; i < bookList.length; i++) {          for (int j = i + 1; j < bookList.length; j++) {
//    to    compare    one    string    with    other    strings                                            if
(bookList[i].getTitle().compareTo(bookList[j].getTitle())  >  0)  {                            //  swapping
tempId = bookList[i].getID();                     tempFirstName = bookList[i].getAuthor_first_name();
tempLastName = bookList[i].getAuthor_last_name();                  tempTitle = bookList[i].getTitle();
tempGenre  =  bookList[i].getGenre();                      tempBorrowed  =  bookList[i].getBorrowed();
//Saves    all    information    stored    in    Position    i    into    an    temporary    array.
bookList[i].setID(bookList[j].getID());
bookList[i].setAuthor_first_name(bookList[j].getAuthor_first_name());
bookList[i].setAuthor_last_name(bookList[j].getAuthor_last_name());
bookList[i].setTitle(bookList[j].getTitle());                    bookList[i].setGenre(bookList[j].getGenre());
bookList[i].setBorrowed(bookList[j].getBorrowed());              //Transfers all Strings from position j
to    Position    i                                                              bookList[j].setID(tempId);
bookList[j].setAuthor_first_name(tempFirstName);
bookList[j].setAuthor_last_name(tempLastName);                              bookList[j].setTitle(tempTitle);
bookList[j].setGenre(tempGenre);                                 bookList[j].setBorrowed(tempBorrowed);
//Finishes the swap.          }        }      }      // print output array      System.out.println(        "--
** LIST OF BOOKS by Title in ALPHABETICAL ORDER ** -- ");       ListBookPosition(bookList);    }    //
List all books sorting by title in alphabetical order.    public String BookIdByTitle(String BookTitle)
throws IOException{       int i = 0;       Book[] booklist = FileLoader();       while(i < booklist.length){
if(booklist[i].getTitle().equalsIgnoreCase(BookTitle)){             return booklist[i].getID();}          i++;
}   return "Book Not found";}}//Returns a BookId That contains the title informed.


public class BorrowData {       private String transactionID;    private String readerID;       private
String bookID;       private Date Drented;    private String Srented;       private Date Dreturned;
private String Sreturned;       private String Next;       SimpleDateFormat formatter = new
SimpleDateFormat("dd/MM/yyyy");      Date date = new Date();              public String
getTransactionID() {       return transactionID;    }    public String getReaderID() {       return readerID;
}    public String getBookID() {       return bookID;    }          public String getSrented() {       return
Srented;    }    public String getSreturned() {       return Sreturned;    }             public void
setTransactionID(String transactioID) {       this.transactionID = transactioID;    }    public void
setReaderID(String readerID) {       this.readerID = readerID;    }    public void setBookID(String
bookID) {       this.bookID = bookID;    }       public void setSrented(String Srented) {       this.Srented =
Srented;    }    public void setSreturned(String Sreturned) {       this.Sreturned = Sreturned;    }
public BorrowData[] Loadborrows() throws FileNotFoundException, IOException{       FileReader file
= new FileReader("BORROW_DATA.csv");       BufferedReader br = new BufferedReader(file);
String read = null;       String[] data;       BorrowData[] load = new BorrowData[200];       int i = 0;
while((read = br.readLine())!=null)       {               data = read.split(",");       //Initializes the String
array with all information in one line, splitting by line.             //------------             BorrowData
borrow = new BorrowData();       borrow.setTransactionID(data[0]);
borrow.setReaderID(data[1]);       borrow.setBookID(data[2]);       borrow.setSrented(data[3]);
//writes into an borrow all information stored from an line.             try{
borrow.setSreturned(data[4]);          } catch(ArrayIndexOutOfBoundsException a){
System.out.println("Book Not returned");          borrow.setSreturned("no");          load[i] = borrow;
return load;          //In case the array doesn't have an 5th space, catch the exception and return the
array we have so far.       }                 load[i] = borrow;       i++;    }                 return load;    }
//Load all borrowals from the BORROW_DATA csv file and stores into an array of BorrowData to
further usage.       public boolean BookIsBorrowed(BorrowData[] borrowlist, String bookId){
int i = 0;       while(i < borrowlist.length){          System.out.println("Reading: " +borrowlist[i]);
try{             if(borrowlist[i].getBookID().equals(bookId) &&
borrowlist[i].getSreturned().equals("no")){             return true;            }}
catch(NullPointerException npe){             System.out.println("End of List");             return false;
}          i++;}          return false;    }    // Check BORROW_DATA csv for an Id that contains the Id
```

```
informed, return true if the book was found and has returned status as "no".      // Returns false if
the book wasn't found or if the returned status is diferent than "no"        public String
NextTransaction(BorrowData bd) throws IOException{          int i = 0;      int next;
BorrowData[] load = bd.Loadborrows();      //Store the BorrowData array into an array, so It is read
just once before the loop.          while(i < load.length){      System.out.println("reading " + load[i]);
try        {        if(!load[i].getTransactionID().isEmpty()){          next =
(Integer.valueOf(load[i].getTransactionID()) + 1);          Next = String.valueOf(next);}
//If the reading is not empty, Changes the string into an Int and stores in a int variable adding one
into it, this would be the next transaction ID.        }      catch (NullPointerException npe){
System.out.println("End Of List");        //It will find the end of the list whenever it reaches a null
pointer exception, this way we know the value next is the next ID in the list.        return Next;
}      i++;        }      return Next;                  }   //Checks the borrowal data file and return the
next int value for a new transaction ID.        public void WriteBorrowal(BorrowData bd, String
ReaderId, String bookID) throws IOException{          FileWriter fw = new
FileWriter("BORROW_DATA.csv", true);      fw.append(bd.NextTransaction(bd)+",");
fw.append(ReaderId +",");      fw.append(bookID +",");      fw.append(formatter.format(date) + ",");
fw.append("no \n");      fw.flush();      fw.close();        //Inserts the line into the next available line
in the document, and saves it after flushing the stream.        System.out.println("new Transaction
Added successfully");        //inform the user the transaction was successful.        }   //Writes a
borrow into the Csv file.      public void WriteReturn(Reader reader, Book[] BkList, BorrowData bd,
BorrowData[] bdList, BookLoader Bl) throws IOException{              FileWriter fw = new
FileWriter("BORROW_DATA.csv", false);      int i = 0;      while(i < bdList.length ){
fw.append(bdList[i].getTransactionID() +",");      fw.append(bdList[i].getReaderID() +",");
fw.append(bdList[i].getBookID() +",");      fw.append(bdList[i].getSrented() + ",");
if(!bdList[i].getSreturned().equals("no")){          //If the value stored in Sreturned is not "no" it will
be a date, meaning the book was returned,        fw.append(bdList[i].getSreturned() + "\n");}
//knowing the book was returned, writes the date of return into the csv File.        else{
fw.append("no \n");                    //knowing the book wasn't returned, append "no" into the
return column.        }      }      fw.flush();      fw.close();        //Inserts the line into the next
available line in the document, and saves it after flushing the stream.        System.out.println("new
Transaction Added successfully");        //inform the user the transaction was successful.        }
//Overwrites the Borrowing file with the updated information in memory.      public void
ReturnBook(String bookId, BorrowData[] bdList){          int i = 0;          while(i < bdList.length){
if(bdList[i].equals(bookId) && bdList[i].getSreturned().equals("no")){
bdList[i].setSreturned(formatter.format(date));              System.out.println("Book written into
Memory");      }else{      i++;}      }            }      @Override   public String toString() {      return
"BorrowData{" + "transactionID=" + transactionID + ", readerID=" + readerID +          ", bookID=" +
bookID + ", Srented=" + Srented + ", Sreturned=" + Sreturned + '}';                    }}
```

```
public class Interactor {    private BufferedReader read = new BufferedReader(new
InputStreamReader(System.in)); //reader used to collect user input.    private String choice; //String
to store buffered data to be used in other menus.    private String buffer; //String to store buffered
data to be used in other menus, secondary.    private int menu; // integer stores the menu item
chosen after parsed into int.            public void Welcome(Reader reader, Book[] BkList, BorrowData
bd, BorrowData[] bdList, BookLoader Bl) throws IOException{System.out.println  ("*---------------------
------------------------------------------*\n" + "|            Welcome to the Easy Library System
|\n" + "|       Please type the options in the interactive menu        |\n" + "|
|\n" + "|    1: Create new Reader                         |\n" + "|    2: Borrowings
|\n" + "|    3: Search                                |\n" + "|    4: Queues
|\n" + "|    5: History                               |\n" + "|    6: ShutDown
|\n" + "*-----------------------------------------------------------------*"            );     choice =
read.readLine();System.out.println("You typed: " + choice);            if(IsNumber(choice)){
//If checks if the input can be converted into an int.        menu = Integer.valueOf(choice);
switch(menu){            case 1:              System.out.println("You've selected: NEW READER\n ");
NewReader(reader, BkList, bd, bdList, Bl);              //case one informs the user the option selected
and open the respective method/menu              break;            case 2:
System.out.println("You've selected: BORROWINGS\n");              Borrowings(reader, BkList, bd,
bdList, Bl);              //case two informs the user the option selected and open the respective
method/menu              break;            case 3:              System.out.println("You've selected:
SEARCH");              SearchMenu(reader, BkList, bd, bdList, Bl);              //case three informs the
user the option selected and open the respective method/menu              break;            case 4:
System.out.println("You've selected: QUEUES");              Queues(reader, BkList, bd, bdList, Bl);
//case four informs the user the option selected and open the respective method/menu
break;            case 5:              System.out.println("You've selected HISTORY");
History(reader, BkList, bd, bdList, Bl);                          //case five informs the user the option
selected and open the respective method/menu              break;                        case 6:
System.out.println("Bye!!!");              System.exit(0);                              default:
System.out.println("Please select a number from the menu\n");              Welcome(reader, BkList,
bd, bdList, Bl);              //case default informs the user the option isn't one option and open this
menu again.         }         } else{         System.out.println("Just type the number, try again\n");
//End of if one, the input is not a number, opens this menu again.         Welcome(reader, BkList, bd,
bdList, Bl);}                      }//dispays welcome message and instructs the user how to navigate
through the Menu.//each case sends the user to a new different menu with instructions//Validate
inputs if it's a number and if the number is in the menu.//////////// FIRST MENU ITEM    public void
NewReader(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl)
throws IOException{    System.out.println("What would you like to do?\n"              + "1: Go Back
\n"              + "2: Add new Reader       \n");    choice = read.readLine();
System.out.println("You typed: " + choice);    //Collect user input    if(IsNumber(choice)){      //If to
check if the input was a number, If it was a number it continues      menu = Integer.valueOf(choice);
switch(menu){         case 1:           Welcome(reader, BkList, bd, bdList, Bl);           //Goes back to
the menu.           break;         case 2:           System.out.println("Please type the reader name
\n");           choice = read.readLine();              System.out.println("You typed: " + choice);
reader.setName(choice);           System.out.println("Please type the reader email \n");
choice = read.readLine();              System.out.println("You typed: " + choice +"\n");
reader.setEmail(choice);              System.out.println("Are these information correct? \n"
+ "Name: "+ reader.getName()+"    \n"                          + "Email:"+ reader.getEmail() + " \n"
+ "1: Yes                   \n"                          + "2: No                   \n");              choice =
read.readLine();                    System.out.println("You typed: " + choice);
if(IsNumber(choice)){                              //If to check if the input was a number, If it was a number
it continues                           menu = Integer.valueOf(choice);                          switch(menu){
case 1:                              System.out.println("Attempting to add new User with new Id: "+
reader.NextReader(reader));                               reader.WriteReader(reader, reader.getName(),
reader.getEmail());                            Welcome(reader, BkList, bd, bdList, Bl);
//case one write the reader information into the csv file and sends the user back to the welcome
menu.                          break;                          case 2: NewReader(reader, BkList, bd, bdList, Bl);
//case two resets the method so the user can try again.                            break;
default: System.out.println("Please select a number from the menu\n");
NewReader(reader, BkList, bd, bdList, Bl);}//end of switch two, sends the user back to the previous
menu in case default                              }                              else{
System.out.println("Just type the number, try again\n");                              NewReader(reader,
BkList, bd, bdList, Bl);}                              //end if/else 2, for number typed, not a number sends the
user back to the beggining of this method.               default: System.out.println("Please select a
number from the menu\n");               NewReader(reader, BkList, bd, bdList, Bl);               //End of
switch one, sends the user back in case default activated.                     }       }        else{
System.out.println("Just type the number, try again \n");       NewReader(reader, BkList, bd, bdList,
Bl);}    //end if/else 1, for number typed, not a number sends the user back to the beggining of this
method.}//Interaction to add new user to the CSV readers file.//Colects user name and email, checks
for duplicates, and writes the csv file.// also options to go back and confirm if information is correct
before submit.// validates data entered in the menu interaction is correct and asks for correction if
not./////////// SECOND MENU ITEMSpublic void Borrowings(Reader reader, Book[] BkList,
BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{
System.out.println("What would you like to do?            \n"                  + "1: Register a book to a user
\n"                  + "2: Register a book back to the Shelves  \n"                  + "3: Go back
\n"); choice = read.readLine();       System.out.println("You typed: " + choice);if(IsNumber(choice)){
//If checks if the value typed is an integer, if so, proceeds the menu interaction    menu =
Integer.valueOf(choice);    switch(menu){       case 1:           BookToUser(reader, BkList, bd, bdList,
Bl);           //Directs the user to the menu for registering  a book to an user.           break;       case 2:
ReturnBook(reader, BkList, bd, bdList, Bl);           //Directs the user to the method for returning a
book to the shelves.           break;       case 3:           Welcome(reader, BkList, bd, bdList, Bl);
//Directs the user back to Main menu.           break;              default: System.out.println("Please
select a number from the menu\n");       Borrowings(reader, BkList, bd, bdList, Bl);}       //end of
switch two, sends the user back to the previous menu in case default    }else{
System.out.println("Just type the number, try again \n");       Borrowings(reader, BkList, bd, bdList,
Bl);}       //end if/else, for number typed, not a number sends the user back to the beggining of this
method.}//Menu directs the user to three options: Borrow a book to a reader, Return a book to the
shelves or Go back to Main menu.// Validates data entered in the menu interaction before
proceeding, informing the user what was typed, so they can correct typping.public void
BookToUser(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl)
throws IOException{       System.out.println("Please chose one option              \n"                  + "1:
```

```
Book by Title            \n"            + "2: Book by Id            \n"            + "3: Go
Back            \n");    choice = read.readLine();        System.out.println("You typed: " +
choice);    if(IsNumber(choice)){       menu = Integer.valueOf(choice);       //Check if the value
informed is actualy an number to proceed with the menu.       switch(menu){          case 1:
System.out.println("Please inform the Title:");            buffer = read.readLine();
System.out.println("You typed: " + buffer);            System.out.println("is the title correct?        \n"
+ "Title: "+ buffer +"        \n"                + "1: Yes              \n"                + "2:
No              \n");            choice = read.readLine();                System.out.println("You
typed: " + choice);                    if(IsNumber(choice)){                    menu =
Integer.valueOf(choice);                    //Check if the value informed is actualy an number to
proceed with the menu.                    switch(menu){                    case 1:
System.out.println("Checking the book in memory");
if(bd.BookIsBorrowed(bdList, BkList[Bl.BookPositionByTitle(buffer)].getID())){
System.out.println("This book Is rented at the momment            \n"
+"Please try again Later or chose another book      \n"                        +"If you want
to be on queue, please go to main menu \n");                    BookToUser(reader, BkList, bd,
bdList, Bl);                    }                    //Verify if the book informed is rented, If so
directs the user back to this method to chose more options                    else{
System.out.println("inform reader ID");                        choice = read.readLine();
System.out.println("Writting Borrowal");
if(reader.ReaderExists(choice)){                        bd.WriteBorrowal(bd, choice,
BkList[Bl.BookPositionByTitle(buffer)].getID()); // Write in BORROW_DATA.csv
BkList[Bl.BookPositionByTitle(choice)].setBorrowed("yes"); // Modify the array Book[]
Bl.FileWriter(BkList);//Write the modification on BOOK_LIST.csv
Welcome(reader, BkList, bd, bdList, Bl);                        }else{
System.out.println("This User Id isn't in our Database  \n"                        +
"Please Register in Main menu      \n"                        +"Directing you to Main
menu...        ");                    Welcome(reader, BkList, bd, bdList, Bl);  //Directs the User
back to main menu                    }                    }//end Case 1
break;                    case 2: System.out.println("Directing you back to the previous menu");
BookToUser(reader, BkList, bd, bdList, Bl);                        break;
default: System.out.println("Please select a number from the menu\n");
BookToUser(reader, BkList, bd, bdList, Bl);                        }
//End of switch 2                    }                    //If the number in the menu isn't a number,
this else is reached.                    else{                    System.out.println("Just type the
number, try again\n");                    BookToUser(reader, BkList, bd, bdList, Bl);
}                    break;        case 2:        System.out.println("Please try again. \n");
BookToUser(reader, BkList, bd, bdList, Bl);            //Case the title informed is wrong, Directs the
user back to try again.        break;            case 3:        System.out.println("Directing user
back to Main Menu");        Welcome(reader, BkList, bd, bdList, Bl);        default:
System.out.println("Please select a number from the menu\n");            BookToUser(reader,
BkList, bd, bdList, Bl);    }      //End of Switch one}// End of If one, ask the user to try again.    else{
System.out.println("Please select a number from the menu\n");    BookToUser(reader, BkList, bd,
bdList, Bl);}}//Method Register a book to a user and uses uther methods to check data and write
data into csv files for Borrowals and BookList.   public void ReturnBook(Reader reader, Book[] BkList,
BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{
System.out.println("Please inform the book Title.");    choice = read.readLine();
System.out.println("You typed: " + choice);    int position = Bl.BookPositionByTitle(choice);
if(position == 0){        System.out.println("Book Not Found, Directing User to Borrowing Menu");
Borrowings(reader, BkList, bd, bdList, Bl);}    //For position zero, means no book was found with this
Id.    //Therefore the user will be sent back to previous Menu.        buffer = BkList[position].getID();
//Information validated, this is a real book Id.    if(bd.BookIsBorrowed(bdList, buffer)){        //If the
book informed shows as borrowed in the BookList, proceeds the method.        Queue qn =
reader.ReaderQueue(buffer);        //Loads queue to verify is there is any        if(qn.getSize() != 0){
//if The queue is zero, it transfers the book to the new Reader                    String nextReader =
qn.RemoveLast();            //remove user from the queue and saves it's ID to a String.
qn.RemoveReaderFromQueue(buffer);            //Remove the user from from Csv file
qn.QueueRefresh();            //Refresh Queue csv File.            bd.WriteBorrowal(bd, nextReader,
buffer);            //append new Borrowal to the book        }else{        //If not queue, proceed here
BkList[position].setBorrowed("no");        //Apply the modification into the book Array as Not
borrowed.        bd.ReturnBook(buffer, bdList);        //Apply the modification into the Memory array as
Not borrowed        bd.WriteReturn(reader, BkList, bd, bdList, Bl);        //Overwrite the csv file with the
updated information.        System.out.println("Return successful, returning to Main menu...");
Welcome(reader, BkList, bd, bdList, Bl);}    }else{        System.out.println("This book wasn't borrowed,
Please check the information typed and try again. \n"                    +  "You will be directed to
Borrowing Menu...                            \n");        Borrowings(reader, BkList, bd, bdList, Bl);        //in
case the information was incorrect, the user will be directed to the borrowing menu.    }}//Verify by
book title the ids into csv files, modifies the files with new information and directs the user to the
main menu// --- to add: queue automatic borrow --- ( possible to send an e-mail to next user
informing the book is available)// validates the information and redirects the user to same menu if
any typing error//THIRD MENU ITEMpublic void SearchMenu (Reader reader, Book[] BkList,
BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{
System.out.println("Please select one option of the menu    \n"                +"1: Search a Book
\n"                +"2: List Books                    \n"                +"3: Search a Reader                \n"
+"4: List Readers                    \n"                +"5: Go Back                        \n");        choice =
read.readLine();            System.out.println("You typed: " + choice);    //Collect user input
if(IsNumber(choice)){        //If to check if the input was a number, If it was a number it continues
menu = Integer.valueOf(choice);        switch(menu){                    case 1:
SearchAbook(reader, BkList, bd, bdList, Bl);            //Directs the user to the Search a book Menu.
break;                    case 2:            ListBooks(reader, BkList, bd, bdList, Bl);            //Directs the
user to the List a book Menu.            break;        case 3:            SearchAreader(reader, BkList, bd,
bdList, Bl);            //Directs the user to Single reader research                    break;        case 4:
ListReaders(reader, BkList, bd, bdList, Bl);            //Directs the user to List readers            break;
case 5:            Welcome(reader, BkList, bd, bdList, Bl);            //Directs the user back to Main
menu.                            default:                    break;                    }//End of First
switch            }else{        System.out.println("Just type the number, try again \n");
SearchMenu(reader, BkList, bd, bdList, Bl);}        //end if/else 1, for number typed, not a number
sends the user back to the beggining of this method.        }//Load SearchMenu        public void
SearchAbook(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl)
throws IOException{        System.out.println("Would you like to search a book by: \n"                + "1:
```

```
Search by title        \n"        + "2: Search by Author and title     \n"        + "3: Go
Back.          \n");    choice = read.readLine();      System.out.println("You typed: " +
choice);  //Collect user input   if(IsNumber(choice)){    //If to check if the input was a number, If it
was a number it continues     menu = Integer.valueOf(choice);    switch(menu){          case
1: System.out.println("Please type the book title \n");      choice = read.readLine();
System.out.println("You typed: " + choice);      //Collect user input
System.out.println("---- This is your book details ---- \n" +
BkList[Bl.BookPositionByTitle(choice)].toString());        SearchMenu(reader, BkList, bd, bdList, Bl);
//Displays the book details and return to search menu.            break;          case
2:      System.out.println("Please type the book title \n");      choice = read.readLine();
System.out.println("You typed: " + choice);          System.out.println("Please type the author
first name \n");      buffer = read.readLine();        System.out.println("You typed: " + buffer);
System.out.println("Please type the author last name \n");      String temp = read.readLine();
System.out.println("You Typed:         \n"            +  "Book Title: "+ choice +"     \n"
+  "Author first name: "+ buffer+"\n"              +  "Author last name: "+ temp +  "\n" );
int position = Bl.BookPositionTitleAndAuthor(choice, buffer, temp);       //Checks if the book
mathes any combination in the book array.       //If it doesn't I will return a zero.
if(position != 0){          System.out.println("---- This is your book details ---- \n" +
BkList[position].toString());          SearchMenu(reader, BkList, bd, bdList, Bl);         //inform
the user the book details, and redirect to Search menu.        }else{
System.out.println("Book not found, redirecting you to the previous menu");
SearchAbook(reader, BkList, bd, bdList, Bl);         //inform the user the book wasn't found, and
redirect to this menu        }          break;           case 3:
SearchMenu(reader, BkList, bd, bdList, Bl);       //Redirect the user back to the previous menu.
break;          default:        System.out.println("This is not a menu option, try again \n");
SearchAbook(reader, BkList, bd, bdList, Bl);             break;           }//End of First
switch        }else{    System.out.println("Just type the number, try again \n");
SearchAbook(reader, BkList, bd, bdList, Bl);}    //end if/else 1, for number typed, not a number
sends the user back to the beggining of this method.      }//Searches a single book whether by title
or a combination of title and author name.//Writes the book details in the terminal if
found.//Validates users input.public void ListBooks(Reader reader, Book[] BkList, BorrowData bd,
BorrowData[] bdList, BookLoader Bl) throws IOException{     System.out.println("Would you like to
List all book by: \n"         + "1: Position in Memory        \n"         + "2: Alphabetic
Author's name     \n"         + "3: Alphabetic Title          \n"         + "4: Go Back
\n");    choice = read.readLine();      System.out.println("You typed: " + choice);  //Collect user
input   if(IsNumber(choice)){    //If to check if the input was a number, If it was a number it
continues     menu = Integer.valueOf(choice);    switch(menu){     case 1:
Bl.ListBookPosition(BkList);        System.out.println("You are being redirected to Search Menu...
\n");        SearchMenu(reader, BkList, bd, bdList, Bl);       //List all books in memory to the
user and directs them back to the search menu        break;          case 2:
Bl.ListBookAuthor(BkList);        System.out.println("You are being redirected to Search Menu...
\n");        SearchMenu(reader, BkList, bd, bdList, Bl);       //List all books in memory sorting
by author's name and directs them back to the search menu             break;
case 3:       Bl.ListBookTitle(BkList);        System.out.println("You are being redirected to
Search Menu... \n");        SearchMenu(reader, BkList, bd, bdList, Bl);            break;
case 4:            SearchMenu(reader, BkList, bd, bdList, Bl);          //Redirects user to the search
menu.          break;                    default: System.out.println("Wrong number, please choose
a number from the menu \n");          ListBooks(reader, BkList, bd, bdList, Bl);          break;     }
}else{      System.out.println("Just type the number, try again \n");      ListBooks(reader, BkList, bd,
bdList, Bl);}      //end if/else 1, for number typed, not a number sends the user back to the beggining
of this method.}//Menu offers option to List all books in memory, To sort in alphabetical order and
list, or by author's alphabetical order also.public void SearchAreader(Reader reader, Book[] BkList,
BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{
System.out.println("Would you like to search a Reader by: \n"              + "1: Search by id
\n"              + "2: Search by name              \n"              + "3: Go Back.                    \n");
choice = read.readLine();          System.out.println("You typed: " + choice);    //Collect user input
if(IsNumber(choice)){      //If to check if the input was a number, If it was a number it continues
menu = Integer.valueOf(choice);      switch(menu){              case 1: System.out.println("Please
type the reader ID \n");          choice = read.readLine();              System.out.println("You typed: " +
choice);          //Collect user input          Reader[] readerList = reader.ReaderLoader();
System.out.println("---- This is your reader details ---- \n" );
reader.ReaderDetailsById(choice);                    SearchAreader(reader, BkList, bd, bdList, Bl);
//Displays the reader details and return to search menu.                    break;                    case
2:          System.out.println("Please type the Name \n");          choice = read.readLine();
System.out.println("You typed: " + choice);
System.out.println(reader.ReaderDetailsbyName(choice));          //Displays the reader details and
return to search menu.          SearchAreader(reader, BkList, bd, bdList, Bl);                    break;
case 3:            SearchMenu(reader, BkList, bd, bdList, Bl);          //Redirect the user back to the
previous menu.          break;              default:          System.out.println("This is not a menu
option, try again \n");          SearchAbook(reader, BkList, bd, bdList, Bl);                    break;
}//End of First switch          }else{      System.out.println("Just type the number, try again \n");
SearchAbook(reader, BkList, bd, bdList, Bl);}      //end if/else 1, for number typed, not a number
sends the user back to the beggining of this method.          }//Menu offers option to search a Reader
by ID or by name.//Validates the input and returns communication if any problem.public void
ListReaders(Reader reader, Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl)
throws IOException{      System.out.println("Would you like to List all readers by: \n"              +
"1: ID                    \n"              + "2: Name, Alphabetically          \n"              + "3: Go
Back                    \n");      choice = read.readLine();          System.out.println("You typed: " +
choice);   //Collect user input   if(IsNumber(choice)){      //If to check if the input was a number, If it
was a number it continues      menu = Integer.valueOf(choice);      switch(menu){          case 1:
reader.ListReaderByPosition(reader);          System.out.println("You are being redirected to Search
Menu... \n");          SearchMenu(reader, BkList, bd, bdList, Bl);          //List all books in memory
to the user and directs them back to the search menu          break;              case 2:
reader.ListReaderbyName(reader);          System.out.println("You are being redirected to Search
Menu... \n");          SearchMenu(reader, BkList, bd, bdList, Bl);          //List all books in memory
sorting by author's name and directs them back to the search menu                    break;
case 3:            System.out.println("Sending you back to Search Menu... \n");
SearchMenu(reader, BkList, bd, bdList, Bl);                    //Redirects user to the search menu.
break;                    default: System.out.println("Wrong number, please choose a number from
the menu \n");          ListReaders(reader, BkList, bd, bdList, Bl);          break;     }   }else{
```

```
System.out.println("Just type the number, try again \n");       ListReaders(reader, BkList, bd, bdList,
Bl);}       //end if/else 1, for number typed, not a number sends the user back to the beggining of this
method.       }//Load ListReader Menu//FOURTH MENU ITEMpublic void Queues(Reader reader,
Book[] BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{
System.out.println("Welcome to the Queue menu, please choose the according options            \n"
+  "1: Enter in a Queue for a Book                                   \n"                 +  "2: Check the next on
queue for a book                                   \n"                 +  "3: Go back.
\n");       choice = read.readLine();           System.out.println("You typed: " + choice);    //Collect user
input    if(IsNumber(choice)){       //If to check if the input was a number, If it was a number it
continues       menu = Integer.valueOf(choice);       switch(menu){              case 1:
System.out.println("Please inform the Book title. \n");              choice = read.readLine();
System.out.println("You typed: " + choice);              buffer = choice;
if(BkList[Bl.BookPositionByTitle(choice)].getBorrowed().equals("no")){
System.out.println("The book is not rented and doens't have a queue \n"                            + "You
will be directed to the Borrowing menu... \n");                 Borrowings(reader, BkList, bd, bdList, Bl);
//If the book found states as NOT rented, the user is directed to borrowal menu.          }else{
if(BkList[Bl.BookPositionById(choice)].getBorrowed().equals("yes")){
System.out.println("Please inform the User who wants to enter in Queue for this book. \n");
choice = read.readLine();                          System.out.println("You typed: " + choice);
reader.AppendToQueue(buffer, choice);                    System.out.println("Reader added, The book
Queue for this book is "+ reader.ReaderQueue(buffer) + "\n");
System.out.println("You'll be redirected to Main Menu...");                    Welcome(reader, BkList, bd,
bdList, Bl);}                    }                              break;                     case 2:
System.out.println("Please inform the Book Title: ");             choice = read.readLine();
System.out.println("You typed: " + choice);
System.out.println(reader.ReaderQueue(Bl.BookIdByTitle(choice)).RemoveLast()+ "Is the Next in
Line.");              System.out.println("You're being redirected to Main menu...");
Welcome(reader, BkList, bd, bdList, Bl);             break;                     case 3:
System.out.println("You're being redirected to Main menu...");             Welcome(reader, BkList, bd,
bdList, Bl);              break;                     default: System.out.println("Wrong number, please choose
a number from the menu \n");             Queues(reader, BkList, bd, bdList, Bl);             break;
}}else{       System.out.println("Just type the number, try again \n");      Welcome(reader, BkList, bd,
bdList, Bl);}       //end if/else 1, for number typed, not a number sends the user back to the beggining
of this method.}//Load Queues Menu//FIFTH MENU ITEMpublic void History(Reader reader, Book[]
BkList, BorrowData bd, BorrowData[] bdList, BookLoader Bl) throws IOException{    int i = 0;
System.out.println("Would you like to List          \n"                 + "1: All History                    \n"
+ "2: Reader History                   \n"                 + "3: Go Back                          \n");       choice =
read.readLine();           System.out.println("You typed: " + choice);    //Collect user input
if(IsNumber(choice)){       //If to check if the input was a number, If it was a number it continues
menu = Integer.valueOf(choice);    switch(menu){          case 1:
System.out.println("Printing History List: \n \n");             while(i< bdList.length){               try{
System.out.println("Transaction Id : " + bdList[i].getTransactionID());
System.out.println("Reader Id : " + bdList[i].getReaderID());               System.out.println("Date
Rented : " + bdList[i].getSrented());               System.out.println("Date Returned : " +
bdList[i].getSreturned());               System.out.println("Book Id: : " + bdList[i].getBookID());
System.out.println("---------------------------------------- \n");                    i++;        }
catch(NullPointerException e){               System.out.println("END OF LIST, Redirecting user to Main
Menu... \n \n \n");               Welcome(reader, BkList, bd, bdList, Bl);                  }}
System.out.println("END OF LIST, Redirecting user to Main Menu... \n \n \n");
Welcome(reader, BkList, bd, bdList, Bl);              break;                        case 2:
System.out.println("Please inform Reader Id... \n");              choice = read.readLine();
System.out.println("You typed: " + choice);              //Collect user input                          i = 0;
System.out.println("Printing History List for userId:" + choice + " \n \n");              while(i <
bdList.length){                  try{                  if(bdList[i].getReaderID().equals(choice)){
System.out.println("Transaction Id : " + bdList[i].getTransactionID());
System.out.println("Reader Id : " + bdList[i].getReaderID());                  System.out.println("Date
Rented : " + bdList[i].getSrented());                  System.out.println("Date Returned : " +
bdList[i].getSreturned());                  System.out.println("Book Id: : " + bdList[i].getBookID());
System.out.println("Transaction Id : " + Bl.BookIdByTitle(bdList[i].getBookID()));
System.out.println("---------------------------------------- \n");                  }}
catch(NullPointerException e){                                  }                                i++;              }
System.out.println("END OF LIST, Redirecting user to Main Menu... \n \n \n");
Welcome(reader, BkList, bd, bdList, Bl);                                                break;
case 3:               System.out.println("Redirecting user to Main Menu... \n \n \n");
Welcome(reader, BkList, bd, bdList, Bl);              break;                                  default:
System.out.println("Wrong number, please choose a number from the menu \n");
History(reader, BkList, bd, bdList, Bl);}     }else{        System.out.println("Just type the number, try
again \n");       History(reader, BkList, bd, bdList, Bl);}        //end if/else 1, for number typed, not a
number sends the user back to the beggining of this method.}//Load history Menu//TOOLpublic
boolean IsNumber(String choice){           if(choice.isEmpty()){          return false;   }             try{
int i = Integer.parseInt(choice);    } catch (NumberFormatException nfe) {    return false;}    return
true;                  }    //Checks if the String collected in Choice is a number.           }
```

```
public class Queue {       private QueueNode first;    private QueueNode last;    private int size;
String[][] temp;       public Queue(){       this.first = null;       this.last = null;       this.size = 0;    }
//Constructor for a brand new Queue.       public void Add(QueueNode data){       QueueNode node
= new QueueNode(data.toString());       if (size == 0){          last = node;          first = node;
size++;}else{       //In case this is the list is empty             node.setNext(first);       first = node;
size++;}       //If it's the first element added, the first and the last are the same.    }    //Add a new
element as first position          private QueueNode NodePosition(int position){            if(size <= 0 ||
position > (size - 1)){          System.out.println("Invalid Position");          return null;       }
//Validates the position to avoid any pointer exception.             QueueNode currentNode = first;
int i = 0;             while(i < position){        i++;        currentNode = currentNode.getNext();              }
return currentNode;    }    //Method to find a node whithin the position asked for.             public
String[][] QueueDetails() throws IOException{                String[] temp;         //Temporary array,
Objective is to store an line from the Csv File.                   FileReader file = new
FileReader("QUEUE.csv");          BufferedReader br = new BufferedReader(file);
FileReader file2 = new FileReader("QUEUE.csv");          BufferedReader br2 = new
BufferedReader(file2);          String read = null;          //Reader and string for read, data from the line
in the csv file                  int size = 0;                 while(br.readLine() != null){            size++;
}                    String[][] TempArray = new String[size][];          //count the size we need for our array
of arrays from the number          //of lines the csv document has, then create the array with the size
//Resets the reader to the beggining of the document.          int i = 0;                  while(i < size){
temp = br2.readLine().split(",");             //Stores a line in an temporary array.            TempArray[i]
= temp;             //Stores the line array into the array of arrays.            i++;                       }
return TempArray;    }     //This Method returns an array that contains all lines from the Csv file
stored as arrays for further interaction.       private String BookRemovalPosition() throws
IOException{       String[][] Qd = QueueDetails();       String temp;       int i = 0;                  while(i <
Qd.length){                          if(Qd[i].length <= 1){                         //If the book is in the list and
There is no             temp = String.valueOf(i);             return temp;}                 //If the book Was
found, Return the index value for it in the Array.          i++;}                  return "Book Not in The list
Or Book Contains Queue";       }    //This method checks in the Queue Details if and book is empty
and returns the Position as an String.    //String was chosen so an Error message could be used in the
same method, to sabe code-lines.       private void QueueCsvRemoveWrite() throws IOException{
String position = BookRemovalPosition();       int p = Integer.valueOf(position);             String[][] Qd =
QueueDetails();       temp = new String[(Qd.length - 1)][2];       int i = 0;               while(i <
Qd.length){                   if(i != p){                         temp[i][0] = Qd[i][0];            temp[i][1] =
Qd[i][1];                        //when I is different than p, It copies the array normally                }else{
try{            temp[i][0] = Qd[i+1][0];            temp[i][0] = Qd[i+1][0];            i++;            p++;
//When i is equal to p, it writes the next and keep i equals to p until it finds an null exception
}catch(ArrayIndexOutOfBoundsException e){             System.out.println("End of bounds, New Array
Ready. \n");       }                    }             //It writes all in the array until the position is reached.
//Once the position is reached, it ignores the position and writes the nexts.          i++;}             //The
array is ready, It's time to rewrite the Queue:              FileWriter fw = new FileWriter("QUEUE.csv",
false);       i = 0;       while(i < temp.length){          fw.append(temp[i][0] + ",");
fw.append(temp[i][1] + "\n");          i++;       }       fw.flush();       fw.close();
System.out.println("Queue Lists updated successfuly. \n");                      }    //This method removes
a book that doesn't have any Queue and rewrite the Queue File with updated information.
public void RemoveReaderFromQueue(String BookId) throws FileNotFoundException, IOException{
String[][] temp = QueueDetails();              FileWriter fw = new FileWriter("QUEUE.csv", false);
int i = 0;        while(i < temp.length){                    if(temp[i][0].equals(BookId)){          String y =
temp[i][1];           temp[i][1]= y.substring(0, y.length() -2);}           //Remove the last two characters
from the list (an space and the last User Id).           i++;                      fw.append(temp[i][0] + ",");
fw.append(temp[i][1] + "\n");                  }       fw.flush();       fw.close();                         }
public void QueueRefresh() throws IOException{                   try{       QueueCsvRemoveWrite();}
catch(NumberFormatException e){          System.out.println("The list contains only books with
Queue");       }    }    //Encapsulation to just apply the private Method to refresh the Csv File.
//Validates for Format exception that occurs when there is now book without queues.       public
String RemoveLast(){              QueueNode temp = last;              if(size == 0){       return null; }       //If
it's empty there is no queue, returns null                    if(size == 1){       last = null;       first = null;
size--;       return temp.getData();       }       //If there was one item, remove it and return the value of
it.              QueueNode secondLast = NodePosition(size - 2);              secondLast.setNext(null);
last = secondLast;       size--;                 return temp.getData();}    //Removes the last node, and
return the string equivalent to it.          public void AddToQueue(Queue queue, String ReaderId){
QueueNode qn = new QueueNode(ReaderId);       queue.Add(qn);                        }       public int
getSize() {       return size;    }                             }
```

```
public class QueueNode {    private String data;    private QueueNode next;           public
QueueNode(String data){       this.data = data;       this.next = null;       //Constructor of nodes for my
Queue.       //The next node has to be empty.           }    public String getData() {       return data;    }
public QueueNode getNext() {       return next;    }       public void setNext(QueueNode next) {
this.next = next;    }    @Override    public String toString() {       return data;    }          }
```

```
public class Reader {       private String Id;    private String name;    private String email;       Queue q
= new Queue();                  String Next;    int next;    public void setId(String Id) {       this.Id = Id;    }
public String getId() {       return Id;    }    public void setName(String name) {       this.name = name;
}       public String getName() {       return name;    }    public void setEmail(String email) {
this.email = email;    }       public String getEmail() {       return email;    }              public Queue
ReaderQueue(String bookId) throws FileNotFoundException, IOException{       FileReader file = new
FileReader("QUEUE.csv");       BufferedReader br = new BufferedReader(file);       String read = null;
String[] data;              int i = 0;                              while((read = br.readLine()) != null){       data =
read.split(",");       //Colect a line from the csv File, stores in an array.       //Position zero will be the
book Id and position 1 will be my Queue.          String temp;       try{          temp = data[0];
}catch(ArrayIndexOutOfBoundsException e){          //Sometimes the array might be empty, in that
case,          //The loop already went trought the list and If any value matches with          //The book
ID, we have our queue populated by now.          System.out.println("End of List");
return q;       }                        if(data[0].equals(bookId)){             //If the book Id has a queue, We
collect the queue and store in a array.          System.out.println("BookId Has a Queue");
String[] queue = data[1].split(" ");                                  int j = 0;             while(j < queue.length){
QueueNode qn = new QueueNode(queue[j]);             q.Add(qn);             j++;
```

```
//populate my Queue with all nodes containing an String with the reader Id.          }               }
}          return q;          }   //Method returns a NodeList for the specified book,   //The first Item
removed from that list will be the next person in line      public Reader[] ReaderLoader() throws
FileNotFoundException, IOException{      FileReader file = new FileReader("READER_DATA.csv");
BufferedReader br = new BufferedReader(file);      String read = null;      String[] data;      Reader[]
ReaderArray = new Reader[200];      int i = 0;            while((read = br.readLine())!=null)      {
data = read.split(",");      //            //------------      Reader reader = new Reader();
reader.setId(data[0]);      reader.setName(data[1]);      reader.setEmail(data[2]);
ReaderArray[i] = reader;      i++;         }      return ReaderArray;   }   //Writes all readers containing
in the CSV file into a Reader array      public boolean ReaderExists(Reader[] r, String name, String
email) throws IOException{               int i = 0;      while(i < r.length){            try         {
if(r[i].getName().equals(name) && r[i].getEmail().equals(email)){            return true;         }
}      catch (NullPointerException npe){         System.out.println("End Of List");         return false;
}      i++;         }      return false;               }   //Check the user list and return true if user already
exists, or false if no user with combination name and email was found.      public boolean
ReaderExists(String id) throws IOException{      Reader [] Rlist = ReaderLoader();            int i = 0;
while(i < Rlist.length){            try         {         if(Rlist[i].getId().equals(id)){            return true;
}                  }      catch (NullPointerException npe){         System.out.println("End Of List");
return false;                              }      i++;         }                  return false;   }   // Verify if the
reader Id informed is in READER_DATA vsc file, returns false if not found, or true if found.      public
String NextReader(Reader r) throws IOException{      int i = 0;            Reader[] readerList =
r.ReaderLoader();                  while(i < readerList.length){      System.out.println("reading " +
readerList[i]);                  try         {         if(!readerList[i].getId().isEmpty()){            next =
(Integer.valueOf(readerList[i].getId()) + 1);            Next = String.valueOf(next);}
}      catch (NullPointerException npe){         System.out.println("End Of List");         return Next;
}      catch (NumberFormatException nfe){         System.out.println("End Of List");         return
Next;      }            i++;                     }      return Next;   }   //Reads The users Ids we have and
return the next available ID.      public void WriteReader(Reader r, String name, String email) throws
FileNotFoundException, IOException{                  if(ReaderExists(ReaderLoader(), name, email)){
System.out.println("The user already exists, Impossible to register");      }else{            FileWriter fw
= new FileWriter("READER_DATA.csv", true);      fw.append(r.NextReader(r)+",");
fw.append(name+",");      fw.append(email+"\n");      fw.flush();      fw.close();
System.out.println("new User Added successfully");   }   }   //Writes information into the Reader File
and goes to next line.      public void ReaderDetailsById(String id) throws IOException{            int
pointer;            try{      pointer = Integer.valueOf(id);}      catch(NumberFormatException e){
}      if(ReaderExists(id)){         int i = Integer.valueOf(id) -1;      Reader[] readerList =
ReaderLoader();      pointer = i;         System.out.println("Reader Name: " +
readerList[pointer].getName()+". \n"                     + "Reader E-Mail "+
readerList[pointer].getEmail()+ ". \n"                     +  "Reader Registered number: "+
readerList[pointer].getId() + ". \n");            }      else{         System.out.println("READER NOT
FOUND, PLEASE CHECK YOUR TYPPING OR REGISTER.");      }         }   //Writes all Readers Details
Sorting it by ID.   public String ReaderDetailsbyName(String name) throws IOException{
String Statement = " **User not Found** ";      int i = 0;      Reader[] readerList = ReaderLoader();
while(i < readerList.length){                  try{         if(readerList[i].getName().equals(name)){
Statement = "Reader Name: " + readerList[i].getName()+". \n"                  + "Reader E-Mail "+
readerList[i].getEmail()+ ". \n"                    +  "Reader Registered number: "+ readerList[i].getId() +
". \n";                          return Statement;           }}          catch(NullPointerException e){
}                              i++;       }                          return Statement;    }    //Check if the user is in the
csv file, returns an String with different results depending on if the user was found or not.       public
void ListReaderByPosition(Reader reader) throws IOException {          Reader[] Rl = ReaderLoader();
int i = 0;       while(i < Rl.length){                    try{          System.out.println("Reader Name: " +
Rl[i].getName()+". \n"                       + "Reader E-Mail "+ Rl[i].getEmail()+ ". \n"                      +
"Reader Registered number: "+ Rl[i].getId() + ". \n");}          catch(NullPointerException e){
}                              i++;       }                 }    //Prints all readers by position in the Csv file.   public
void ListReaderbyName(Reader reader) throws IOException {            String tempId;       String
tempName;       String tempEmail;              Reader[] Rl = ReaderLoader();                   for (int i = 0; i
< Rl.length; i++) {           for (int j = i + 1; j < Rl.length; j++) {                         // to compare one string
with other strings              try{             if (Rl[i].getName().compareTo(Rl[j].getName()) > 0) {
// swapping                 tempId = Rl[i].getId();                tempName = Rl[i].getName();
tempEmail = Rl[i].getEmail();                 //Saves all information stored in Position i into an temporary
array.                           Rl[i].setId(Rl[j].getId());                Rl[i].setName(Rl[j].getName());
Rl[i].setEmail(Rl[j].getEmail());                 //Transfers all Strings from position j to Position i
Rl[j].setId(tempId);                  Rl[j].setName(tempName);                 Rl[j].setEmail(tempEmail);
//Finishes the swap.              }}             catch(NullPointerException e){                          }
}       }              // print output array       System.out.println(          "-- ** LIST OF READERS IN
ALPHABETICAL ORDER ** -- ");       ListReaderByPosition(reader);                 }    //Resorts the Reader
array and print it, now in alphabetical order.       public void AppendToQueue(String BookId, String
UserId) throws IOException{              String[][] Queue = q.QueueDetails();             int i = 0;
boolean oldQueue = false;                              while(i < Queue.length){
if(Queue[i][0].equals(BookId)){              Queue[i][1] += " "+UserId;          oldQueue = true;}
i++;}              if(oldQueue){          FileWriter fw = new FileWriter("QUEUE.csv", false);          i = 0;
while(i < Queue.length){              fw.append(Queue[i][0] + ",");              fw.append(Queue[i][1] +
"\n");              i++;          }          fw.flush();          fw.close();}       //If This was a book already in the
list, append the Reader to the queue.       else{          FileWriter fw = new FileWriter("QUEUE.csv",
false);          i = 0;          while(i < Queue.length){              fw.append(Queue[i][0] + ",");
fw.append(Queue[i][1] + "\n");              i++;}              //Writes all the other Queues
fw.append(BookId + ",");               fw.append(UserId + " \n");              //Writes a new Queue for the
New book.       }                }   //Appends user to an BookQueue, or create a new Queue for a book If
there is none                 @Override    public String toString() {       return "Reader{" + "Id=" + Id + ",
name=" + name + ", email=" + email + '}';    }                  }
```

Conclusion

The Final work has much space for improvement of tools, and small corrections for the data shown. The system created makes these changes not difficult with the tools created inside the code.

The system has as biggest failure the Null Pointer Exceptions, as the arrays are the main tool used to store and interact with data. And as these structures mutates often, The array size does as well.
It’s not always the same solution implemented into every exception, but basically the most common types of error for this code would be Format, array out of bounds and null exceptions.

This would be a cheap way to implement an offline system for an library to manage their business and the structure gives much space for improvement without modifications on the csv files.
It won’t demand much memory work, therefore the system can expand much more before needs a structural modification.