Development Report

Student	Asher Rashid
ID	U0018369
Module	Object Oriented Programming (CIS4037)
Course	MSc

Note for Students

This Development Report is based on Book Shop System. The report is intended as an example for students to adapt for their ICA Task8 Report.

Total Word count is 1784 words, but when headings, captions, tabular data, table of contents, references and front page are excluded the word count falls below 1600 words, which is within 10% of word limit.

Table of Contents

1. Discussion of GUI

1.1 Overview

In transforming the Console version to a GUI, some components remained unchanged or had minor changes, which Table 1 specifies

Table 1. Use of Console version components in GUI Version

Console Component	Purpose	Change for GUI Version
`Book.java`	Data Class	No change
`bookList`	Class level ArrayList to hold Book objects	No change
`delimiter`	Class level variable, which is separator in data files	No change
`loadBooks()`	Main class method which reads input file to populate array list	Input file is in a sub directory named 'files'
`saveBooks()`	Main class method which processes array list to write data to output file	Output file is in a sub directory named 'files'

1.2 GUI

Figure 1 shows the created GUI, noting it includes components which are used by tasks discussed by other sections of this report.

The main components used are

- JFrame which is container for GUI and is the main class
- JTable named bookTable, used to list book objects
- TextArea named console, used to display information to user
- JLabel named imageLabel, used to display an mage for a selected book
- Buttons to view al books, search for a book and quit application

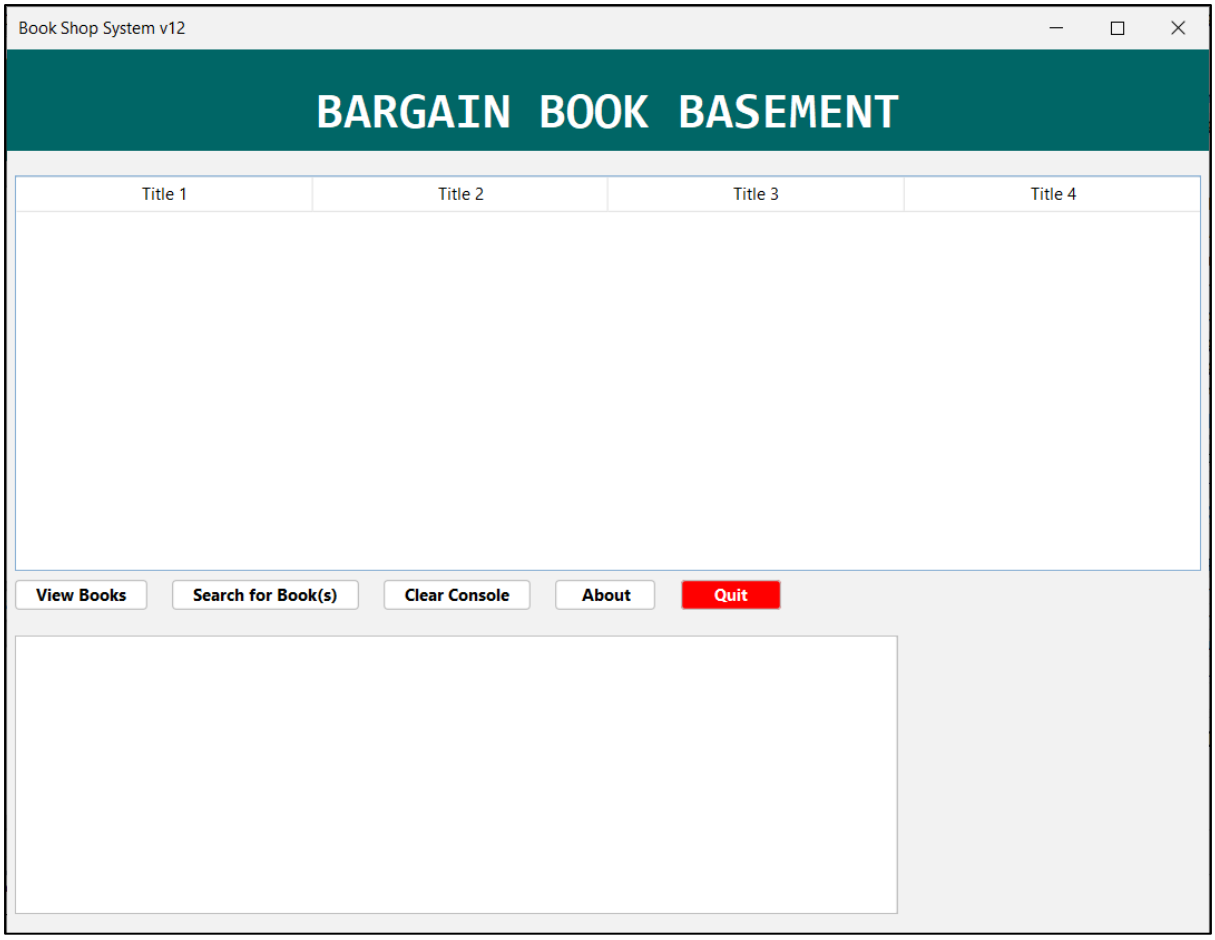


Figure 1: GUI for Book Shop System

Note image Label occupies bottom left of the GUI.

1.3 Abstract Table Model

JTables are basic components that rely on a model, which will be linked to the JTable within the main class.

The model class used is named BookModel and it extends AbstractTableModel abstract class. Figure 2 shows header and fields of the class.

```java
public class BookModel extends AbstractTableModel{

    //fields
    //empty string array for column names
    private String [] columnNames;

    //empty two dimensional object array for data
    private Object[][] data;
```

Figure 2: Header and fields of BookModel class

The start of the Constructor is shown is Figure 3. The parameters are a String Array used to set the number of columns for the table model and a generic ArrayList containing data loaded from the input file.

```java
    //provide a custom constructor
    public BookModel(final String [] colNames, final ArrayList<Book> dataList) {
        //get length of array parameter
        int columnNamesLength = colNames.length;

        //copy parameter array into column names
        columnNames = Arrays.copyOf(colNames, columnNamesLength);
        //get size of arraylist
        int rowLength = dataList.size();

        //set size of data array
        data = new Object[rowLength][columnNamesLength];
```

Figure 3: Start of BookModel class constructor

JTables cannot directly display data from an ArrayList. Thus, the ArrayList data will be used to populate the two-dimensional Object array named data within the mdoel. Each row in data will hold an item from the ArrayList. The number of rows is set to be the same as the number of items from the ArrayList and the number of columns will be the same as number of column headers.

Figure 4 shows the start of an enhanced for loop to acquire the values of fields in each item of the Array List

```java
        //set index variables for data row
        int row=0;

        //loop through array list
        for (Book item: dataList){
            //get fields
            String title = item.getTitle();
            String author = item.getAuthor();
            String isbn = item.getISBN();
            Double price = item.getPrice();
            Integer quantity = item.getQuantity();
```

Figure 4: Processing ArrayList parameter

The fields values are used to populate an one-dimensional Object array, which in turn is assigned to the current row of data (see Figure 5).

```java
            //use fields to create object array
            Object [] dataRow = new Object[] {title, author, isbn, price, quantity};

            //copy row data array into current data row
            data[row] = Arrays.copyOf(dataRow, columnNamesLength);

            //increase row index
            row++;
        }
    }// end constructior
```

Figure 5: Populating each row of the data Array

Some inherited methods are not implemented by the abstract base class, and need to be coded in the BookModel class, i.e., overridden as per Figure 6.

```java
    @Override
    public int getRowCount() {
        //give length of first dimension of data
        return data.length;
    }

    @Override
    public int getColumnCount() {
        //give length of scolumn names
        return columnNames.length;
    }

    @Override
    public Object getValueAt(int row, int column) {
        //get object at insection of row and colun in data
        return data[row][column];
    }

    @Override
    public void setValueAt(Object value, int row, int col) {
        data[row][col] = value;
        fireTableCellUpdated(row, col);
    }
```

Figure 6: Overriding of inherited methods

1.4 Main Class code

In the main class a BookModel object is created and set as the model for the JTable, see Figure 7.

```
model = new BookModel(columnNames, bookList);

//link abstract table model to JTable
bookTable.setModel(model);
```

Figure 7: Creating Book Model object

Oddly, it is not possible to set column headers inside the TableModel class. Instead, it has to be done after a BookModel object has been created. Each column is referenced inside a for loop, the header is set for the column using the corresponding element from columnNames. This code is depicted in Figure 8 below.

```
//set column headers in Jtable
for (int col = 0; col < bookTable.getColumnCount(); col++) {
    //reference current column
    TableColumn column = bookTable.getTableHeader().getColumnModel().getColumn(col);

    //set column header
    column.setHeaderValue(columnNames[col]);
}
```

Figure 8: Setting Column Header

1.5 Testing

Figure 9 shows contents of input file.

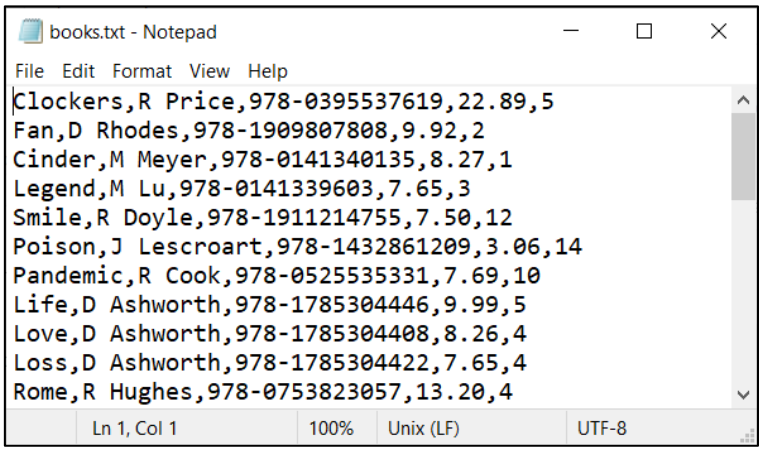
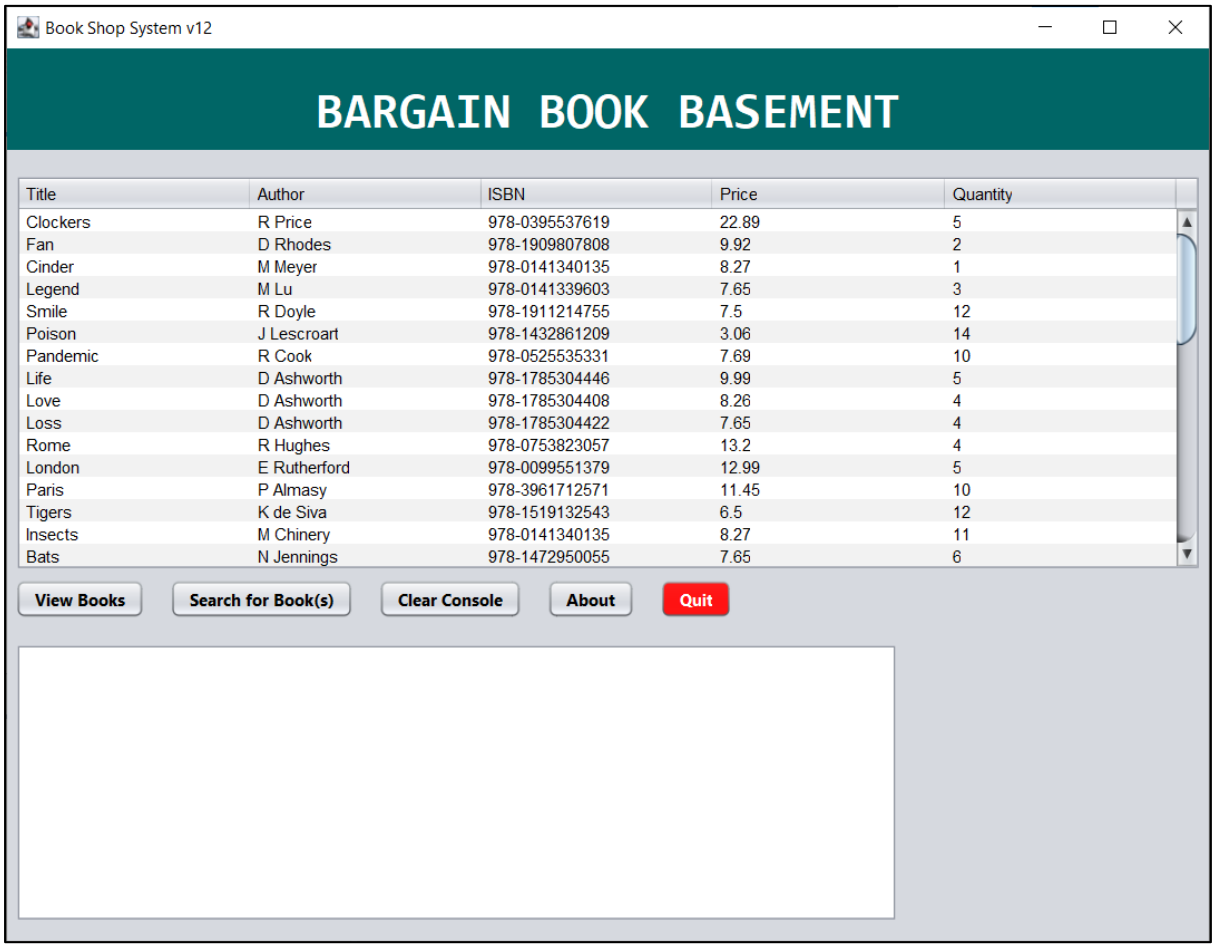


Figure 9: Input data File

Figure 10 depicts the data displayed by the JTable. As can be observed the first lines in data file is displayed as first row in the JTable.



Title	Author	ISBN	Price	Quantity
Clockers	R Price	978-0395537619	22.89	5
Fan	D Rhodes	978-1909807808	9.92	2
Cinder	M Meyer	978-0141340135	8.27	1
Legend	M Lu	978-0141339603	7.65	3
Smile	R Doyle	978-1911214755	7.5	12
Poison	J Lescroart	978-1432861209	3.06	14
Pandemic	R Cook	978-0525535331	7.69	10
Life	D Ashworth	978-1785304446	9.99	5
Love	D Ashworth	978-1785304408	8.26	4
Loss	D Ashworth	978-1785304422	7.65	4
Rome	R Hughes	978-0753823057	13.2	4
London	E Rutherford	978-0099551379	12.99	5
Paris	P Almasy	978-3961712571	11.45	10
Tigers	K de Siva	978-1519132543	6.5	12
Insects	M Chinery	978-0141340135	8.27	11
Bats	N Jennings	978-1472950055	7.65	6

Figure 10: Populated JTable

2. Discussion of Viewing a Book

2.1 Overview

When a row is selected in the JTable, details of a book are shown in the Text area and any associated image displayed in the image label.

2.2 Table events

A JTable row can be selected by the user clicking on a row, i.e., mouse click event. In addition, if a row is already selected it can be changed by the user pressing up and down keys, i.e. key pressed and released events. Figure 11 shows the JTable event methods , all of get index of the selected or newly selected row and pass this integer value to the display book details method().

```
    private void bookTableMouseClicked(java.awt.event.MouseEvent evt) {
        // get the selected row from the table
        int row = bookTable.getSelectedRow();

        //call the method to display details of the book
        displayBookDetails(row);
    }

    private void bookTableKeyPressed(java.awt.event.KeyEvent evt) {
        // get the selected row from the table
        int row = bookTable.getSelectedRow();

        //call the method to display details of the book
        displayBookDetails(row);
    }

    private void bookTableKeyReleased(java.awt.event.KeyEvent evt) {
        // get the slected row from the table
        int row = bookTable.getSelectedRow();

        //call the method to display details of the book
        displayBookDetails(row);
    }
```

Figure 11: JTable Event Methods

2.3 Display Book Details method

From figure 12 it case be seen that `displayBookDetails()` method, first clears the TextArea. After which field values of the book are obtained from the ArrayList using the parameter `index`.

```java
    //method to display singe book details
    public void displayBookDetails(int index){
        //clear console
        console.setText("");

        //get book details
        String title = bookList.get(index).getTitle();
        String author = bookList.get(index).getAuthor();
        String isbn = bookList.get(index).getISBN();
        double price = bookList.get(index).getPrice();
        int quantity = bookList.get(index).getQuantity();
```

Figure 12: Getting field values of a book

The obtained field values are used to construct a message, which is then displayed in the TextArea. The last line of Figure 13 (`console.moveCaretPosition(0);`) moves the insertion point to start of first row; which has the effect of scrolling up the TextArea in case all content cannot be displayed.

```java
        //construct message to display
        String message = String.format("SELECTED BOOK DETAILS%n%n%10s: %s %n%n%10s: %s %n%n%10s: %s %n%n%10s: £ %.2f %n%n%10s: %d",
                "Title", title,
                "Author", author,
                "ISBN", isbn,
                "Price", price,
                "Quantity", quantity
        );

        //display to console
        console.append(message);

        //scroll textarea back to first line
        console.moveCaretPosition(0);
```

Figure 13: Displaying field values of a book

The next section of the method, get the corresponding image filename from the Book item and construct a relative file path from it. The image Label text and icon are cleared.

```
//get image filepath
String imagePath = "files/images/" + bookList.get(index).getImageFileName();

//clear label
imageLabel.setText("");
imageLabel.setIcon(null);
```

Figure 14: Constructing Image file path

Finally, a check takes place of the image file exists and if so the image is set as the icon for the image label. Otherwise, a not available message is displayed in the label.

```
//check if file exists
if ( (new File(imagePath)).exists() ){
    //display image
    imageLabel.setIcon( new ImageIcon( imagePath ) );
} else {
    //set label text
    imageLabel.setText("Image not available.");
}
}
```

Figure 15: Displaying image

2.4 Testing

The JTable row for **London** is selected in the JTable. Figure 16 shows the book's details are displayed in the Text Area and a *not available* message in the Image Label

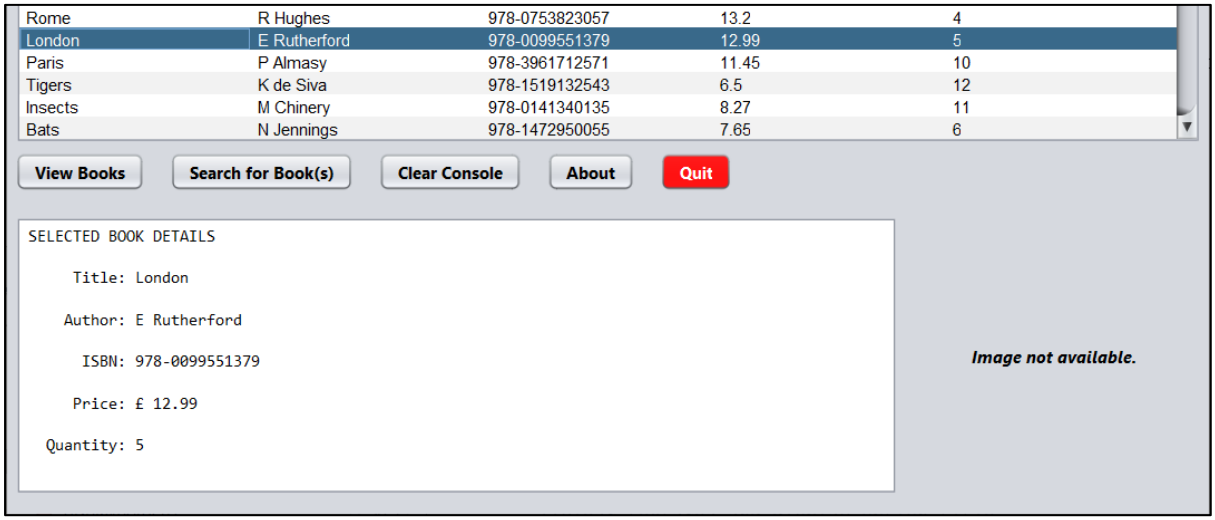


Figure 16: Displaying details of the book titled London

The JTable row for **Paris** is selected in the JTable. Figure 17 shows the book's details are displayed in the Text Area and a image is displayed in the Image Label.

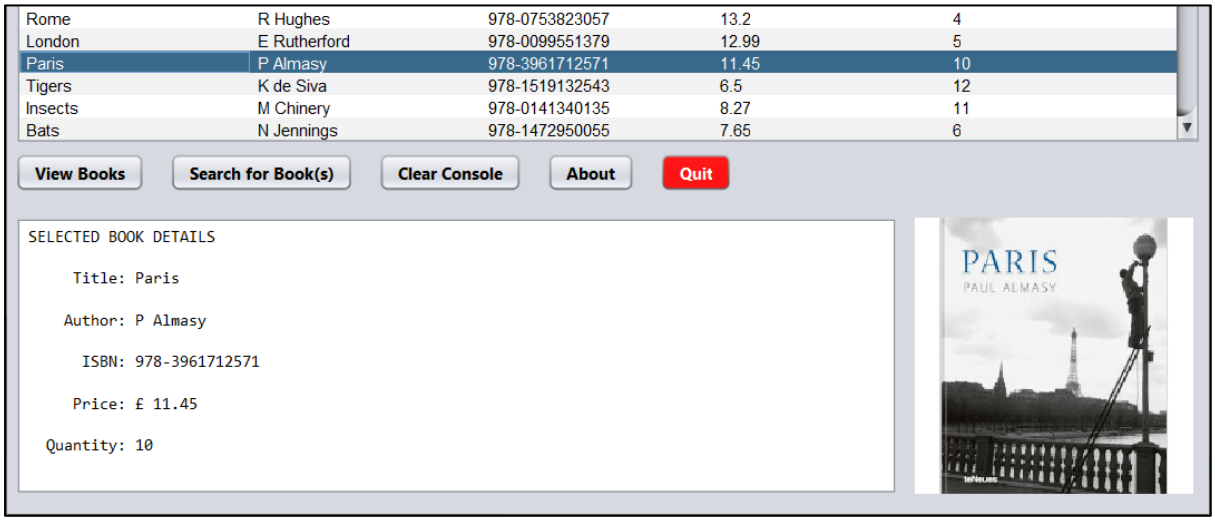


Figure 17: Displaying details of the book titled Paris

3. Discussion of Search Feature

3.1 Overview

The Search feature allows the user to type a search term into a pop-up dialog. This input value is checked to see if it is contained within the the following fields of any Book object

- `title`
- `author`
- `isbn`

If a field produces a match, the details of that Book are displayed to the TextArea.

3.2 Search Button

Figure 11 display the event method code for the Search button.

```
private void searchBookButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //deselct selected table row
    bookTable.clearSelection();

    //empty image label
    imageLabel.setText("");
    imageLabel.setIcon(null);

    //call relevant method
    searchBooks();
}
```

Figure 11: Search button Click event method

The code deselects any selected row in the JTable, clears imageLabel text and or icon. Afterwards the `searchBooks()` method is invoked.

3.3 Search Books method

Figure 12 displays the start code for the Search Book method, which obtains input using a Input dialog.

```java
    //method to organise the search for  book(s) - works with findBooks method
    @SuppressWarnings("null")
    public void searchBooks() {

        //get input
        String target;
        target = JOptionPane.showInputDialog(null,
                "Please enter search data:",
                "Search Book(s) Input",
                JOptionPane.QUESTION_MESSAGE).trim();

        //check for no null or no input
        if (target == null) {
            //user clicked cancelled so return
            return;
        } else if (target.isEmpty() || target.isBlank()) {
            //warn user and return
            JOptionPane.showMessageDialog(null, "No input provided. Please try
            return;
        }
```

Figure 12: Search Book method input code

The provided input is trimmed before being saved. The input variable is checked if it is null, empty, or blank, in which case the method terminates by returning.

```java
        //Forward target to findBooks which will return an ArrayList
        ArrayList<Integer> matchedIndexList = findBooks(target);

        //display title
        console.setText("### SEARCH FOR BOOKS ###\n");
```

Figure 13: Invoking findBooks

If the input is valid the `findBooks` method is called, passing the input value to it (Figure 13). This method returned a array list of integers, each integer is the index of a match book in the Array List bookList.

Figure 14 shows the closing code of the Search Book method. The integer ArrayList is checked if it is empty, which means there were no matches.

```
        //checkif returned array list is empty
        if (matchedIndexList.isEmpty()) {
            //inform user
            console.append("\nNo search results for '" + target +"'.");
        } else {

            //display sub title
            console.append("\nSearch results for '" + target +"'.\n");

            //display header & borders
            displayHeader();
            displayBorder();

            //display book details
            for (int index: matchedIndexList){
                displayBookRow(bookList.get(index));
            }

            //display end border
            displayBorder();
        }

    } //end of method
```

Figure 14: Search Books output code

Otherwise, there are matches which need to be displayed. The integer ArrayList is looped through and the integer value used to get book from ArrayList and is then displayed to the Text Area using the displayBookRow() method.

3.4 Find Book method

In this method, each book in the ArrayList is iterated through. The `title`, `author` and `isbn` is acquired for current book. After which the passed input value (converted to lowercase) is checked if it is contained in any of the three fields. If so the index of the book is added to the indices array list. After the for loop the indices array list is returned.

Figure 15 depicts the code for the `findBooks` method.

```java
    public  ArrayList<Integer> findBooks(String target) {

        //create an Integr Arraylist
        ArrayList<Integer> indices = new ArrayList<>();

        //transform target to lowercase
        target = target.toLowerCase();

        //loop through bookList
        for (int index=0; index<bookList.size(); index++) {

            //get title, author and isbn in lowercase
            String title = bookList.get(index).getTitle().toLowerCase();
            String author = bookList.get(index).getAuthor().toLowerCase();
            String isbn = bookList.get(index).getISBN().toLowerCase();

            //serach within fields for target, check if target is contained within title
            if (title.contains(target)) {
                //if a match add index to arrayList
                indices.add(index);

            //check if target is contained within author
            } else if (author.contains(target)) {
                //if a match add index to arrayList
                indices.add(index);

            //check if target is contained within isbn
            } else if (isbn.contains(target)) {
                //if a match add index to arrayList
                indices.add(index);
            }
        }//end of for loop

        //return arraylist
        return indices;

    } //end of method
```

Figure 16: Find Books method

3.5 Testing

Upon clicking the Search Button and input dialog is displayed into which the input of "`at`" is entered (see figure 17)

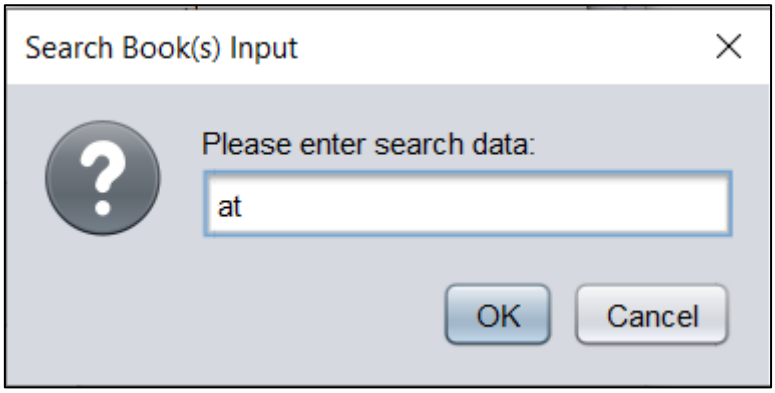


Figure 17: Input dialog

Upon clicking OK button matching books are found and displayed to the Text Area, as per Figure 18.

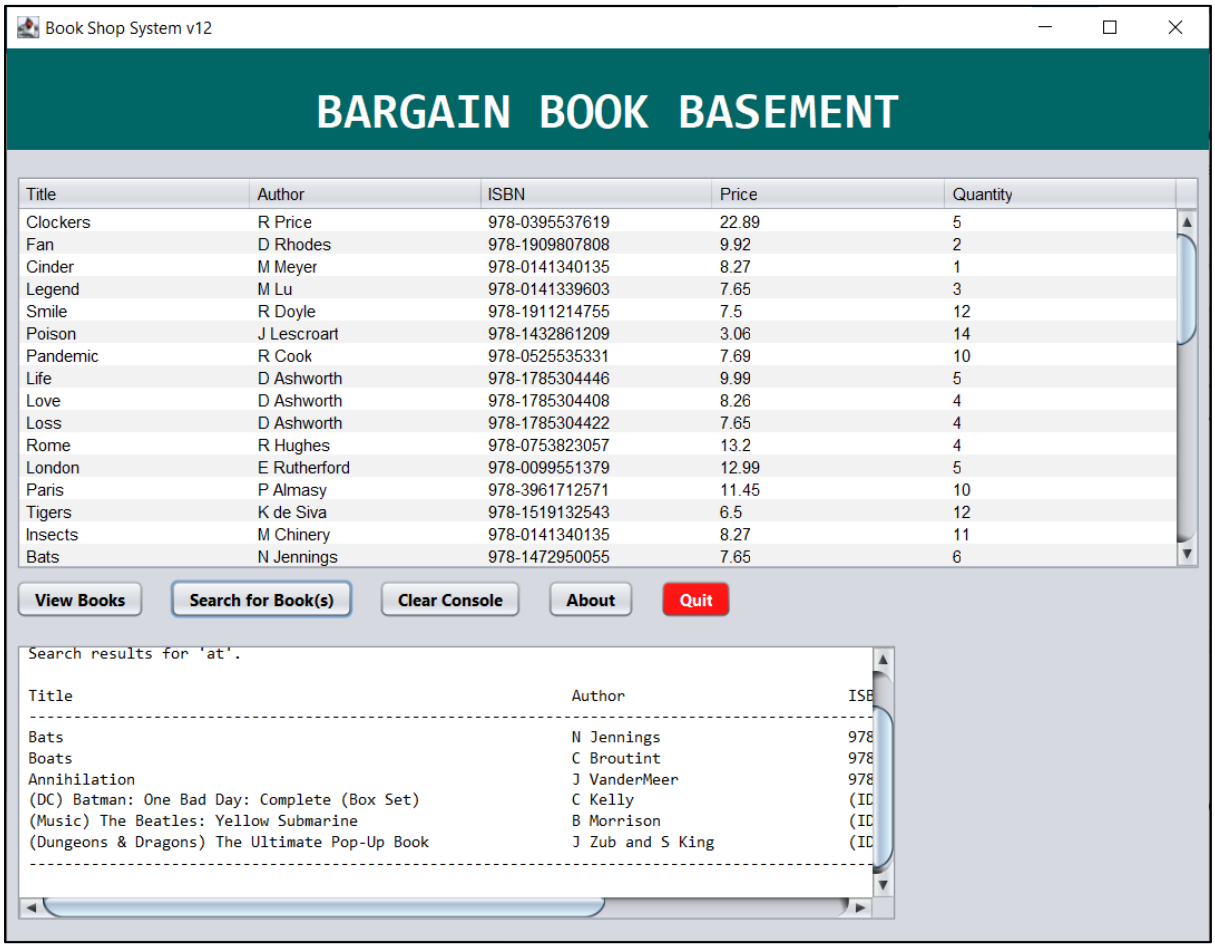


Figure 18: Search Results

Figure 19 shows no results are displayed in the text area, when using an input of "au".

Figure 19: No search results

4. Discussion of Improvement

4.1 Overview

Suggested improvement is to display a hyperlink for the user to cleck when a book has been selected. The hyperlink would be to a search engine, e.g., google.co.uk, to show results for the title of the book.

4.2 Requirements

The hyperlink would be displayed in a JLabel. The foreground colour would be set to blue, and the displayed text underlined.

A class level string variable would be created to store the url for the search

When an row is selected in the JTable, the Display Details would set the displayed text on the hyperlink label and the value of the `url` variable would be set specific for the book title selected in the JTable.

A mouse click event method would be added to the hyperlink label within which the following could be included

```
try {
     Desktop.getDesktop().browse(new URI(url)

} catch (URISyntaxException e) {
     e.printStackTrace();

} catch (Exception e) {
     e.printStackTrace();
}
```

Code Listing 1: Launching web browser to view a URL (Minh, 2019)

The Desktop class is part of `Java.awt` library, the `getDesktop().browse()` method launches default web browser opens the argument as a web page.

5. References

Minh N.H. (2019) *How to create hyperlink with JLabel in Java Swing*. Available at: https://www.codejava.net/java-se/swing/how-to-create-hyperlink-with-jlabel-in-java-swing (Accessed: 01 May 2023)