



**BINUS  
BINUS**

**UNIVERSITY  
INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work )**

**Student Information:**

Name: Rafian Athallah Marchansyah

Student ID: 244004238

**Course Code :** COMP6699 **Course Name :** Object Oriented Programming

**Class :** L2AC **Name of Lecturer(s) :** Jude Joseph Lamug Martinez

**Major :** Computer Science

**Title of Assignment :** The Archive Database  
**Submission**

**Type of                      Pattern**

**Assignment                      :** Final Project

**Due Date :** 22-6-2021 **Submission Date :**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating**

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

Rafian Athallah Marchansyah

## Table of Contents

<b>Project Specification</b>	
.....	<b>3</b>
<b>Solution Design</b>	
.....	<b>3</b>
<b>Program Preview</b>	
.....	<b>5</b>
<b>Code Explanation</b>	
.....	<b>7</b>
<b>Lessons That Have Been Learned</b>	
.....	<b>17</b>
<b>Project Link</b>	
.....	<b>18</b>

# **“The Archive Database”**

**Name: Rafian Athallah Marchansyah**

**ID: 2440042380**

## **I. Project Specification**

### **Purpose:**

Create a GUI database with the help of Java using Swing and MySQL.

### **Audience:**

People who like to collect books and need a way to store and manage what books you have on hand.

### **Aim:**

To create a database that stores information about books including its title, author, genre and price. I will be using the IDE Apache Netbeans to help construct the GUI using Swing and phpMyAdmin to connect our MySQL Database.

### **Packages used:**

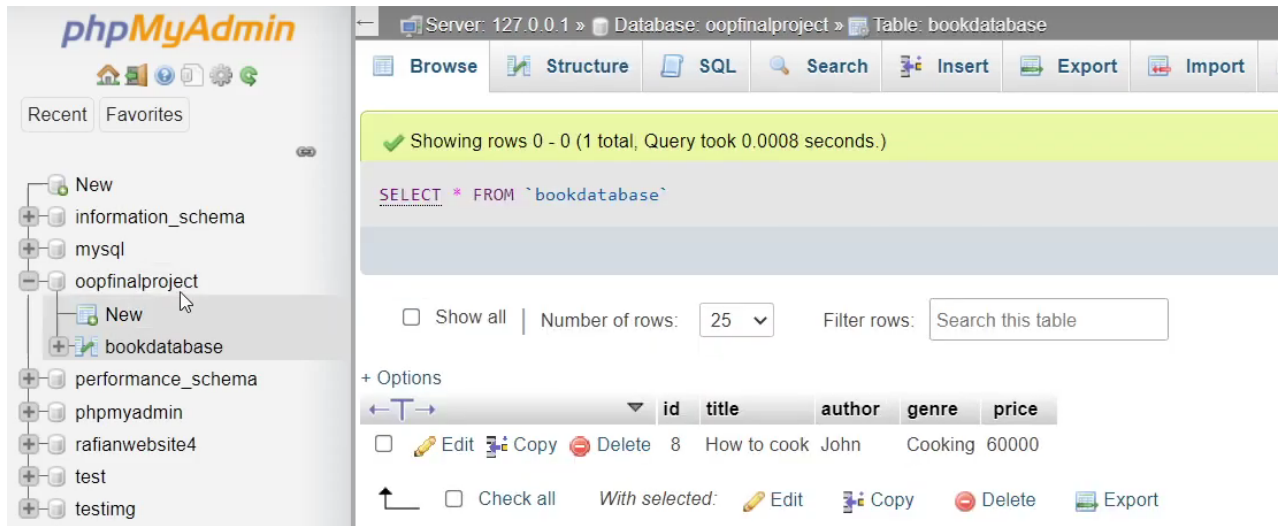
- Java.sql
- Javax.swing
- Java.util.list

### **Libraries used:**

- MySQLconnector/J

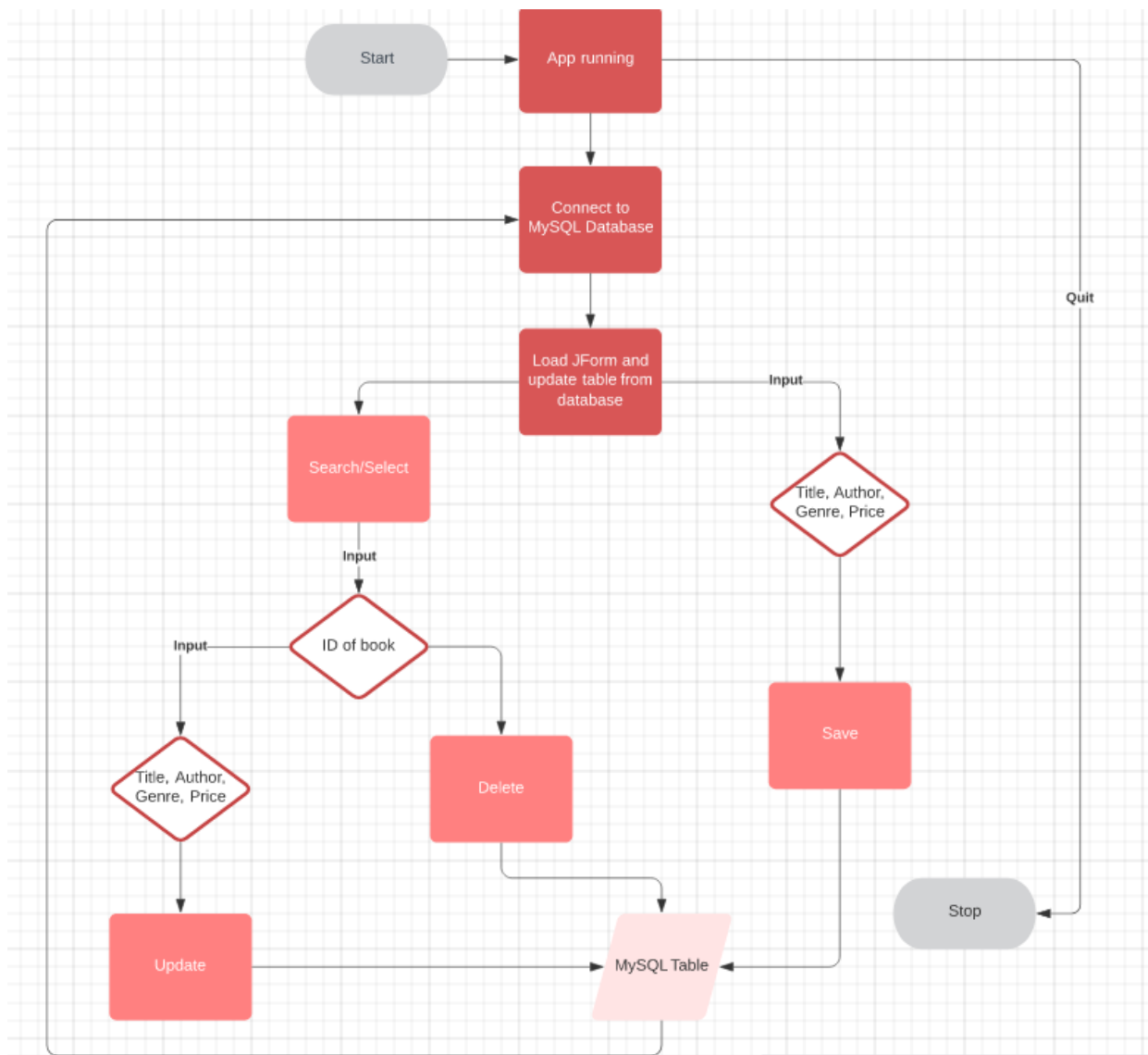
## **II. Solution Design**

The GUI that I have created requires the user to have set up a MySQL database with a table that contains the variables needed for the information of the book (id, title, author, genre, price.)



In this case my database is called “oopfinalproject” which contains the table named “bookdatabase” and each entry in the table has the variables id, title, author, genre and price.

To demonstrate how this MySQL database is connected to our program, here is a simple flowchart:

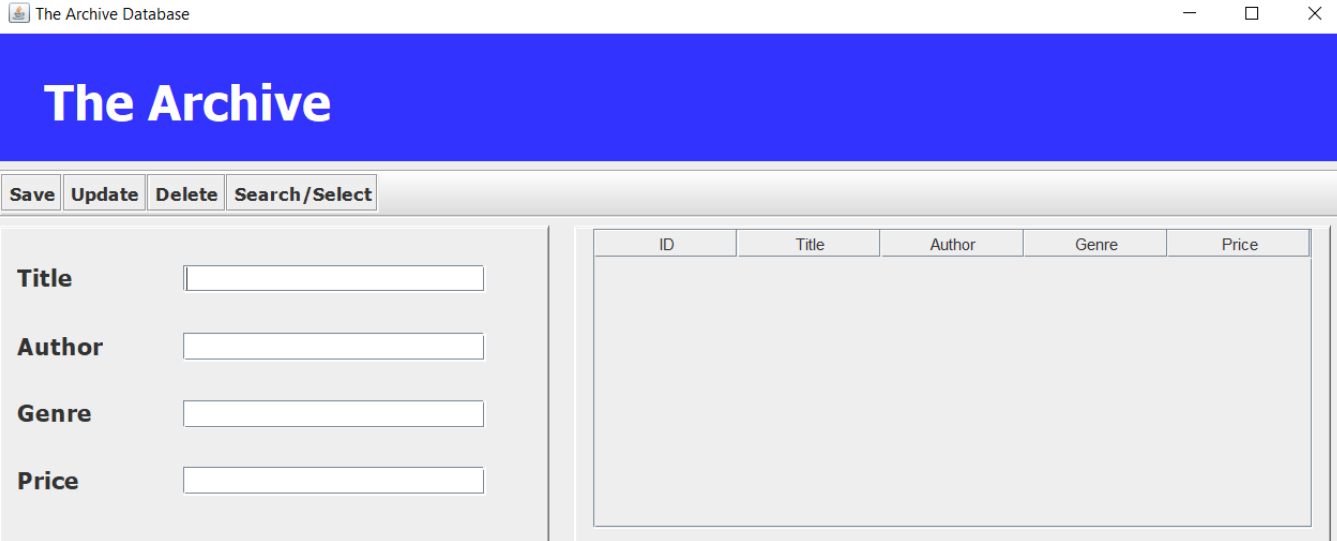


Firstly we need the program to connect to the MySQL database so that we can access it. Then, the user can input the various functions and it will update the MySQL Table which in turn will use the connection that we've established. A better explanation will be in the program preview.

### III. Program preview

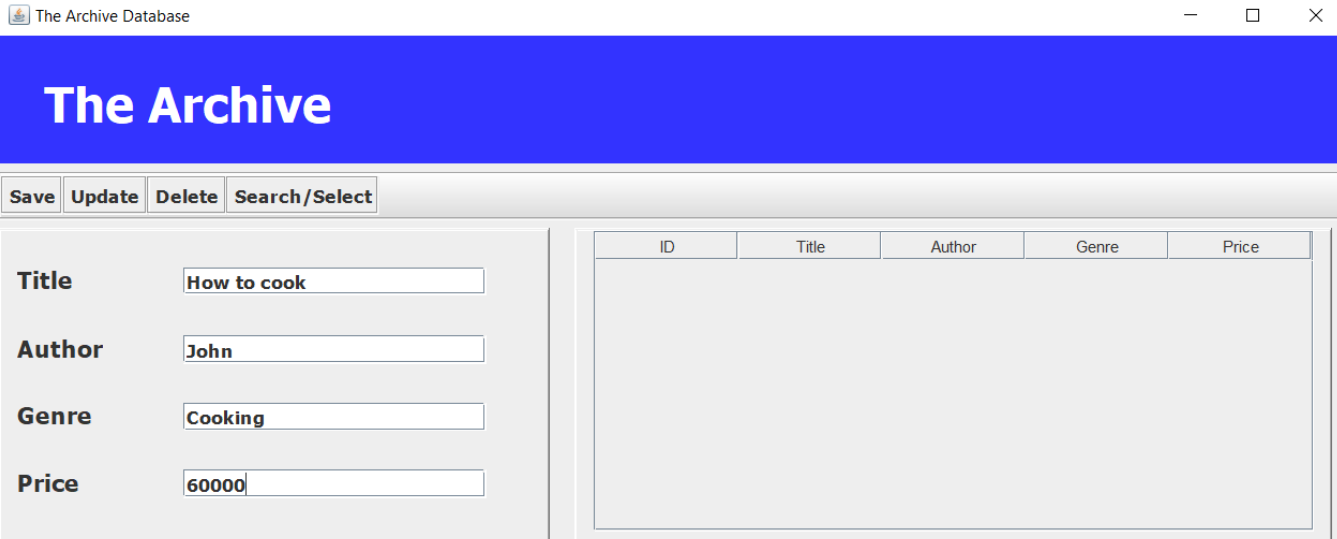
"The Archive Database" is a GUI database that uses Swing and MySQL to store the entries of the database through the use of tables.

When we start the program it will look like this:



The screenshot shows a window titled "The Archive Database". The main header is a blue bar with the text "The Archive" in white. Below the header is a toolbar with four buttons: "Save", "Update", "Delete", and "Search/Select". The main area is divided into two sections. On the left, there are four input fields labeled "Title", "Author", "Genre", and "Price". On the right, there is a table with five columns: "ID", "Title", "Author", "Genre", and "Price". The table is currently empty.

The user can then input the information of the book they want to put in the database:



The screenshot shows the same window as before, but the input fields are now filled with sample data: "Title" is "How to cook", "Author" is "John", "Genre" is "Cooking", and "Price" is "60000". The table on the right remains empty.

And said book will be given an ID (integer) and it will be updated on the MySQL database as well as its information being displayed on the JForm table:

The Archive Database

# The Archive

Save Update Delete Search/Select

Title

Author

Genre

Price

ID	Title	Author	Genre	Price
9	How to cook	John	Cooking	60000

To update/delete an entry, it is crucial that you first select it by clicking the select option which prompts the user to input the ID of the book they want to change/delete, or else it will not work:

The Archive Database

# The Archive

Save Update Delete Search/Select

Title

Author

Genre

Price

Input

Enter ID

9

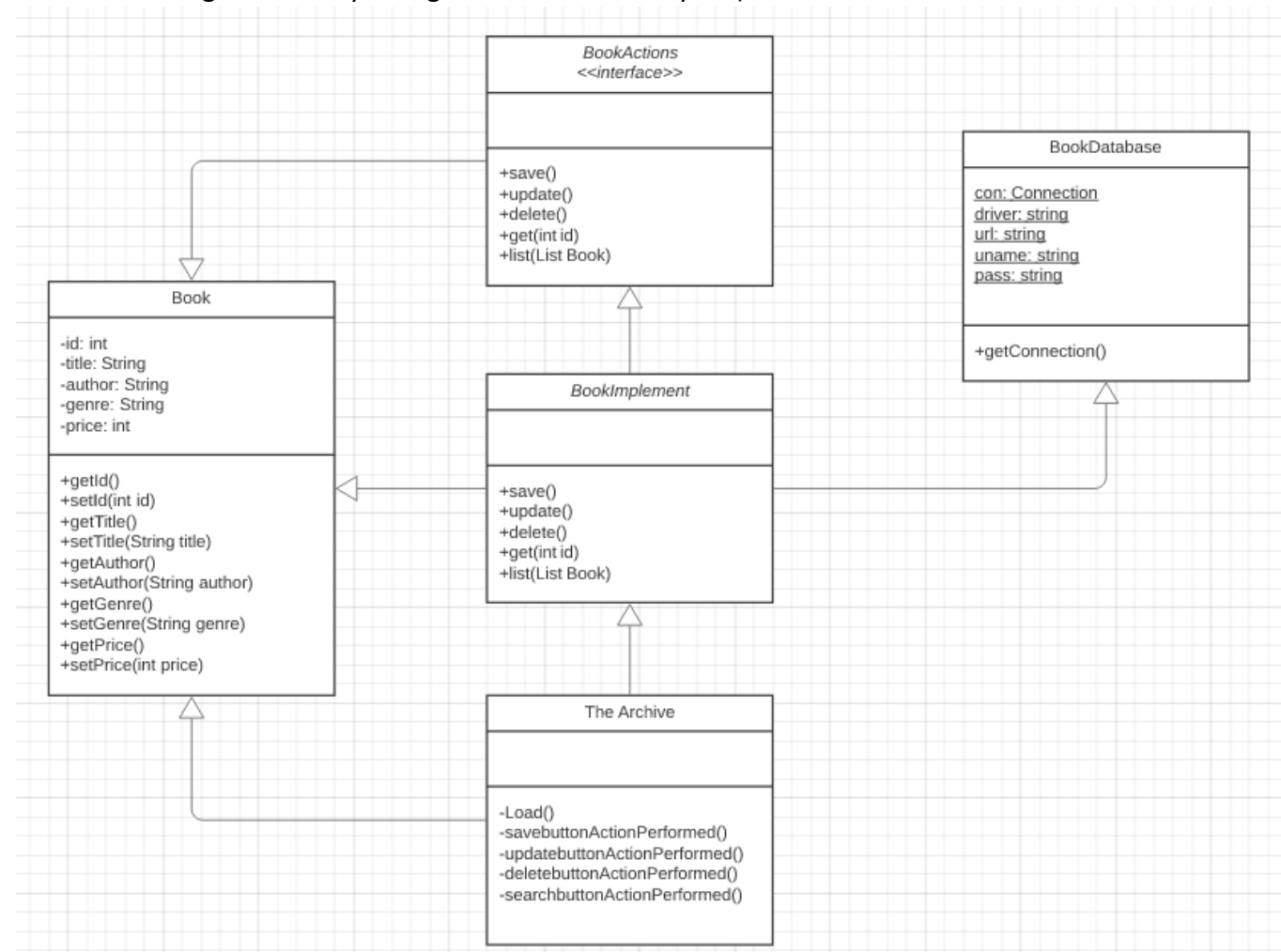
OK Cancel

Author	Genre	Price
John	Cooking	60000

The user can then choose to update the information or delete it which will delete it off the table and the MySQL database.

## IV. Code explanation

This is the UML diagram that showcases roughly how the code work (I've left out the classes that were auto-generated by Swing or wasn't altered by me):



Firstly, we need to establish a connection to the MySQL database with the help of the java.sql package:

```
public class BookDatabase {

    //used to establish connection to the mysql database
    //static is used as it saves memory and this variable will be constant
    //and used throughout the whole code

    static Connection con;
    static String driver = "com.mysql.jdbc.Driver";
    static String url = "jdbc:mysql://localhost/oopfinalproject";
    static String uname = "root";
    static String pass = "";

    public static Connection getConnection() throws Exception{
        if(con == null){
            Class.forName(driver);
            con = DriverManager.getConnection(url,uname, pass);
        }
        return con;
    }
}
```

This is the default class that is used and called whenever a user wants to connect to the MySQL database (in this case program it is when the user saves, selects, updates, & deletes any entries). It inputs the MySQL server details (which is different for every user) in this case mine is hosted on the directory oopfinalproject and my username is the default root and has no password. The getConnection() connects the driver to the server and uses the server details information that has been inputted.

After the connection established, we move on to the book class which is a simple class with variables (ID, author, title, genre, price) that all have setter and getter methods:



```
//getter and setter methods for the variables used for each book (title, author, genre, price)
public class Book {
    private int id;
    private String title;
    private String author;
    private String genre;
    private int price;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }
}
```

We then create an interface which will be implemented through abstract methods later for the save, update, delete, get ID and list method:

```
//interface of the database actions that will be implemented later
public interface BookActions {

    public void save(Book books);
    public void update(Book books);
    public void delete(Book books);
    public Book get(int id);
    public List<Book> list();
}
```

I will now go over each abstract method one by one, starting from save:

```
//implementing interface methods
public class BookImplement implements BookActions{

    @Override
    public void save(Book books) {

        try {
            //calls the connection to the database and takes the inputted title,author,genre,price into the prepared statement
            //and inserts it into the mysql table
            Connection con = BookDatabase.getConnection();
            String sql = "INSERT INTO bookdatabase(title,author,genre,price) VALUES (?, ?, ?, ?)";
            PreparedStatement ps = con.prepareStatement(sql);
            ps.setString(1, books.getTitle());
            ps.setString(2, books.getAuthor());
            ps.setString(3, books.getGenre());
            ps.setInt(4, books.getPrice());
            ps.executeUpdate();
            JOptionPane.showMessageDialog(null, "Saved!");
        } catch (Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Error"); //returns error if not all things are filled
        }
    }
}
```

When save method is called, the program will connect to the MySQL database using the `getConnection()` function that has been explained before, and the program will use the getter methods of the book to get information of the selected book and insert them to the MySQL database table using prepared statements, which replace the empty values with the values given by the getter methods. If all requirements are satisfied, then it will return a successful "Saved!" message, else it will return an error.

Next is the update method, which works almost exactly the same way as the save method:

```

@Override
public void update(Book books) {

    try {
        //same thing as save feature, this time it will update an existing database entry using prepared statements
        Connection con = BookDatabase.getConnection();
        String sql = "UPDATE bookdatabase SET title=?,author=?,genre=?,price=? WHERE id=?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, books.getTitle());
        ps.setString(2, books.getAuthor());
        ps.setString(3, books.getGenre());
        ps.setInt(4, books.getPrice());
        ps.setInt(5, books.getId());
        ps.executeUpdate();

        JOptionPane.showMessageDialog(null, "Updated");
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error");
    }
}

```

After connecting to the MySQL database like before, it will update the values of an existing book using prepared statements again. If all requirements are satisfied, a successful “Updated!” message will pop up.

We then move on to the get method that will help us with searching for a book based on ID:

```

@Override
public Book get(int id) {

    //takes inputted ID to prepared statement and returns the corresponding book information
    Book book = new Book();
    try {
        Connection con = BookDatabase.getConnection();
        String sql = "SELECT * FROM bookdatabase WHERE id=?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        if(rs.next()){

            book.setId(rs.getInt("id"));
            book.setTitle(rs.getString("title"));
            book.setAuthor(rs.getString("author"));
            book.setGenre(rs.getString("genre"));
            book.setPrice(rs.getInt("price"));
        }

    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error");
    }
    return book;
}

```

After connecting to the MySQL database, the method will then search for the entry with the corresponding ID that is inputted with prepared statements, and returns the information of the book with the corresponding ID. If no book is found, it will return an error instead.

Next is the delete method:

```
@Override
public void delete(Book books) {
    //takes the ID of a selected entry and deletes it from the mysql table
    try {
        Connection con = BookDatabase.getConnection();
        String sql = "delete from bookdatabase WHERE id=?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, books.getId());
        ps.executeUpdate();
        JOptionPane.showMessageDialog(null, "Deleted");
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error");
    }
}
```

This one is rather simple, after connecting to the MySQL database it will take the ID (integer) using prepared statements and delete the book entry with the corresponding ID from the MySQL table.

Finally, we have the list method which will help us create the JForm table later:

```

//creating arraylist that stores all the entries of books
@Override
public List<Book> list() {

    List<Book> list = new ArrayList<Book>();
    try {
        Connection con = BookDatabase.getConnection();
        String sql = "SELECT * FROM bookdatabase ";
        PreparedStatement ps = con.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();

        while(rs.next()){
            Book book = new Book();
            book.setId(rs.getInt("id"));
            book.setTitle(rs.getString("title"));
            book.setAuthor(rs.getString("author"));
            book.setGenre(rs.getString("genre"));
            book.setPrice(rs.getInt("price"));

            list.add(book);
        }

    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error");
    }
    return list;
}

```

After connecting to the database, the method will take all information of the entries and create an arraylist containing all the books with their information.

All of these methods will then be used in the JForm class, which I will be explaining next:

```
//loads the book entries from the list into the JForm table
public void Load()

BookImplement bookimplement = new BookImplement();
    List<Book> list = bookimplement.list();
    DefaultTableModel DFT = (DefaultTableModel) jTable1.getModel();
    DFT.setRowCount(0);
    for(Book book: list)
    {
        int id = book.getId();
        String title = book.getTitle();
        String author = book.getAuthor();
        String genre = book.getGenre();
        int price = book.getPrice();
        DFT.addRow(new Object[]{id,title,author,genre,price});
    }
```

To start, we create the load method that will help us load all of the book entries into a JForm table using the list method I have explained before that will then display all the information of the books.

To initiate the JForm, it looks like this:

```
//Loading the UI and table with existing database entries
public TheArchive() {
    initComponents();
    Load();
}
```

initComponents() loads the GUI itself and the load method will update the JForm table to add in the information of all the book entries that are currently in the database.

I will now explain all the code to the buttons found in the JForm GUI, which is the save, update, search/select button, & delete button. For the save button:

```
//save feature
private void savebuttonActionPerformed(java.awt.event.ActionEvent evt) {

    //creates a new instance of Book and sets the inputted variables
    Book book = new Book();
    String title = txttitle.getText();
    String author = txtauthor.getText();
    String genre = txtgenre.getText();
    int price = Integer.parseInt(txtprice.getText());
    book.setTitle(title);
    book.setAuthor(author);
    book.setGenre(genre);
    book.setPrice(price);

    //load the information of the new book into the JForm table
    BookImplement newbook = new BookImplement();
    newbook.save(book);
    Load();
    txttitle.setText("");
    txtauthor.setText("");
    txtgenre.setText("");
    txtprice.setText("");
    txttitle.requestFocus();
}
}
```

When the button is clicked, it will create a new Book instance and set the inputted variables into the corresponding information. It will then save the book using the save method explained before and the JForm table will be updated with the new entry.

```
//search feature
int search;
private void searchbuttonActionPerformed(java.awt.event.ActionEvent evt) {

    //using search method to get information of a book using ID
    //and displays the information of the corresponding book on the text bar
    search = Integer.parseInt(JOptionPane.showInputDialog("Enter ID"));

    BookImplement searchbook = new BookImplement();
    Book book = searchbook.get(search);

    txttitle.setText(book.getTitle());
    txtauthor.setText(book.getAuthor());
    txtgenre.setText(book.getGenre());
    txtprice.setText(String.valueOf(book.getPrice()));
}
}
```

When the search button is clicked, the get method will initiate and search the corresponding book with the inputted ID. it will then set the text fields in the JForm to display the information of the selected book.

```
//update feature
private void updatebuttonActionPerformed(java.awt.event.ActionEvent evt) {

    //same thing as save feature, but this time it updates an existing book
    Book book = new Book();
    String title = txttitle.getText();
    String author = txtauthor.getText();
    String genre = txtgenre.getText();
    int price = Integer.parseInt(txtprice.getText());
    book.setTitle(title);
    book.setAuthor(author);
    book.setGenre(genre);
    book.setPrice(price);
    book.setId(search);

    //updates the new information of the book in the JForm table
    BookImplement updatebook = new BookImplement();
    updatebook.update(book);
    Load();
    txttitle.setText("");
    txtauthor.setText("");
    txtgenre.setText("");
    txtprice.setText("");
    txttitle.requestFocus();
}
```

For the update button, it works exactly the same as the save button, but it updates the book with the corresponding ID with the new information instead, and updates the JForm table with said new information as well.



```
//delete feature
private void deletebuttonActionPerformed(java.awt.event.ActionEvent evt) {

    //using delete method to delete book using inputted id
    //and deletes the book from the JForm table
    Book book = new Book();
    book.setId(search);
    BookImplement deletebook = new BookImplement();
    deletebook.delete(book);
    Load();
    txttitle.setText("");
    txtauthor.setText("");
    txtgenre.setText("");
    txtprice.setText("");
    txttitle.requestFocus();
}
```

Lastly we have the delete button, where it will execute the delete method on the book with the corresponding ID inputted, and delete it from the JForm table as well.

## V. Lessons that Have Been Learned

I have learnt a lot of valuable lessons throughout this project. The most obvious being that I've learned the basics on using Swing, where I learned how to create a GUI with the help of JForm. I have also learned a lot about the MySQL database, and how to implement a table to store data in tandem with my GUI. Setting up the MySQL database and establishing the connection took quite some time to understand, but I'm glad that it worked out in the end. Learning Swing and JForm was also quite interesting to me. The UI creation was fairly simple with the help of Netbeans IDE where I could just design my GUI by dragging and dropping with a lot of customization features. The things I've learned were very basic and my program might look simple, but I've learned quite a lot from it.

Some difficulties I ran into was probably only setting up the MySQL database, as I had to install some external libraries first and learned how to make my program connect to it before trying anything with the JForm.

## **VI. Project link**

<https://github.com/rafianathallah/OOPFinalProject>