



## **OBJECT-ORIENTED VISUAL PROGRAMMING**

### **Final Project Report**



### **Music Studio Rental App (Stacato Studio)**

#### **Group Members:**

Member 1	:	Noelani Aurelia Raintung	CIT – 2023	(001202300115)
Member 2	:	Sarahwati	CIT – 2023	(001202300146)

**BATCH 2023**

**PRESIDENT UNIVERSITY**

Jababeka Education Park, Jl. Ki Hajar Dewantara, Kota Jababeka,  
Cikarang Baru, Bekasi 17550 – Indonesia  
Phone (021) 8910 9762-6, Fax (021) 8910 9768

## **Application Description**

### **1. What does the app do?**

This application is designed for renting band studios. It provides a platform where admin can search for, book, and manage studio rentals based on users specific needs. Some of its features include helping admin keep track, create, update, and delete projects in a simple manner.

### **2. What problem do you solve?**

**Problem:** Musicians often struggle to find studios that meet their specific needs, whether for recording, rehearsals, or mixing and mastering. They need detailed information about the studio's equipment, acoustics, and availability.

**Solution:** The application provides a comprehensive platform where users can search for studios based on specific categories and detailed profiles, ensuring they find the right space for their needs.

**Problem:** Traditional methods of booking studios involve multiple phone calls or emails, leading to delays and potential double bookings. This process is time-consuming and inconvenient.

**Solution:** The application offers a real-time booking system with instant confirmation, making it easy and quick for users to reserve studio time without hassle

.

### **3. What is unique about it?**

**Specialized Studio Categories:** Highly specialized categories such as studios for live band recording, band rehearsal, mixing & mastering, and pre-production.

**Subcategory Options:** Users can search for studios based on subcategories such as equipment needs (drum kit, amplifiers, PA system).

### **4. Why should someone download it**

This application is aimed at individuals with an interest in the music industry, such as musicians and bands, music producers, music agents, music schools, and even music enthusiasts. This application will be very helpful and convenient in every aspect, and it will save a lot of time.

## IMPLEMENTATION OF OOP

### Class

In object oriented programming (OOP), a class is a blueprint or template that defines objects. A class specifies the attributes (variables) and methods (functions) that the objects created from the class will possess.

```
public class Register1 extends javax.swing.JFrame {
    int xx, xy;

    /**
     * Creates new form Register1
     */
    public Register1() throws SQLException {
        initComponents();
        txID.setEnabled(false);
        autonumber();
    }
}
```

### Object

An object is an instance of a class in object-oriented programming (OOP). An object is an entity that has state and behavior. Each object has data (called attributes or properties) and methods (functions) that the object can perform.

```
public class Dashboard extends javax.swing.JFrame {

    private DefaultTableModel tabmode;
    private void tampil_barang(){
        Object[] baris = {"CategoryID", "Name", "Capacity", "MusicEquipment"};
        tabmode = new DefaultTableModel(null, baris);
        t_category.setModel(tabmode);
        String sql = "select * from Category";

        String cari = txtCari.getText();
        try{
            Connection connect = new DBConnect().getConnection();
            Statement stat = connect.createStatement();
            ResultSet hasil = stat.executeQuery(sql);
            while (hasil.next()) {
                String CategoryID = hasil.getString("CategoryID");
                String Name = hasil.getString("Name");
                String Capacity = hasil.getString("Capacity");
                String MusicEquipment = hasil.getString("MusicEquipment");
                String[] data = {CategoryID, Name, Capacity, MusicEquipment};
                tabmode.addRow(data);
            }
            connect.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Menampilkan data GAGAL", "Informasi", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

## Inheritance

Inheritance is a concept in object-oriented programming where a class can inherit properties and methods from another class. The Login class inherits all properties and methods of the javax.swing.JFrame class. This means that all the features and functionalities available in JFrame, such as window management, GUI layout, and event handling, can be used in the Login class without needing to write the code again.

```
public class Login extends javax.swing.JFrame {  
  
    Connection con = null;  
    Statement st = null;  
  
    public Login() {  
        initComponents();  
        try {  
            Class.forName("com.mysql.jdbc.Driver");  
            con = DriverManager.getConnection("jdbc:mysql://localhost/StacatoStudio", "root", "");  
            st = con.createStatement();  
            JOptionPane.showMessageDialog(null, "Connection sucesss.");  
        } catch (Exception ex) {  
            JOptionPane.showMessageDialog(null, "Connection failed." + ex);  
        }  
    }  
}
```

## Polymorphism

The setString() method of the PreparedStatement object can accept various types of parameters, such as String, int, double, and so on. In this code example, it uses the setString() method.

To handle exceptions, it uses the Exception class. However, within the catch block, it catches the specific SQLException separately from Exception.

```
try {  
    Class.forName("com.mysql.cj.jdbc.Driver");  
    Connection DBConnect = DriverManager.getConnection("jdbc:mysql://localhost/StacatoStudio", "root", "");  
  
    String sql = "INSERT INTO Booking (BookingID, CostumersName, StudioID, TimeStart, TimeEnd, TotalPrice) VALUES (?, ?, ?, ?, ?, ?)";  
    PreparedStatement stat = DBConnect.prepareStatement(sql);  
    // Parameter Polymorphism  
    // Metode setString()  
    stat.setString(1, BookingID);  
    stat.setString(2, Name);  
    stat.setString(3, StudioID);  
    stat.setString(4, Start);  
    stat.setString(5, End);  
    stat.setString(6, Price);  
  
    int rowInserted = stat.executeUpdate();  
  
    if (rowInserted > 0) {  
        JOptionPane.showMessageDialog(this, "Task added.");  
        resetForm();  
        getData();  
    }  
} catch (Exception e) {  
    JOptionPane.showMessageDialog(this, "Task can not be added: " + e.getMessage(), "Error", JOptionPane.WARNING_MESSAGE);  
}
```

```

private DefaultTableModel tabmodel;
private void tampil_barang3(){
    Object[]baris = {"BookingID", "CostumersName", "StudioID", "TimeStart", "TimeEnd", "TotalPrice"};
    tabmodel3 = new DefaultTableModel(null, baris);
    t_bkg.setModel(tabmodel3);
    String sql = "select * from Booking";

    // Runtime Polymorphism
    // Penggunaan kelas yang lebih umum (Exception)
    try{
        Connection connect = new DBConnect().getConnection();
        Statement stat = connect.createStatement();
        ResultSet hasil = stat.executeQuery(sql);
        while (hasil.next()) {
            String BookingID = hasil.getString("BookingID");
            String CostumersName = hasil.getString("CostumersName");
            String StudioID = hasil.getString("StudioID");
            String TimeStart = hasil.getString("TimeStart");
            String TimeEnd = hasil.getString("TimeEnd");
            String TotalPrice = hasil.getString("TotalPrice");
            String[]data = {BookingID, CostumersName, StudioID, TimeStart, TimeEnd, TotalPrice};
            tabmodel3.addRow(data);
        }
        connect.close();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Menampilkan data GAGAL", "Informasi", JOptionPane.INFORMATION_MESSAGE);
    }
}
}

```

## Encapsulation

Encapsulation is a concept in object-oriented programming that combines data and methods that operate on the data into a single unit called an object. In the context of the Java code you provided, there are several features that demonstrate the use of encapsulation:

1. **Private Variables:** In this example, there are variables `xx` and `xy` declared outside the constructor with the `private` access modifier. This isolates these variables so that they can only be accessed within the `Register1` class.
2. **Private Methods:** In this example, there might be a `autonumber()` method that is not shown in the code snippet you provided. If this method is only used within the `Register1` class, you might declare it with the `private` access modifier so that it can only be accessed from within the `Register1` class.

By declaring variables and methods as private, you restrict direct access to them from outside the `Register1` class. This allows you to control how data is manipulated and processed within that class, which is an important aspect of the encapsulation concept.

```

//
public class Register1 extends javax.swing.JFrame {
    //encapsulation
    private int xx, xy;

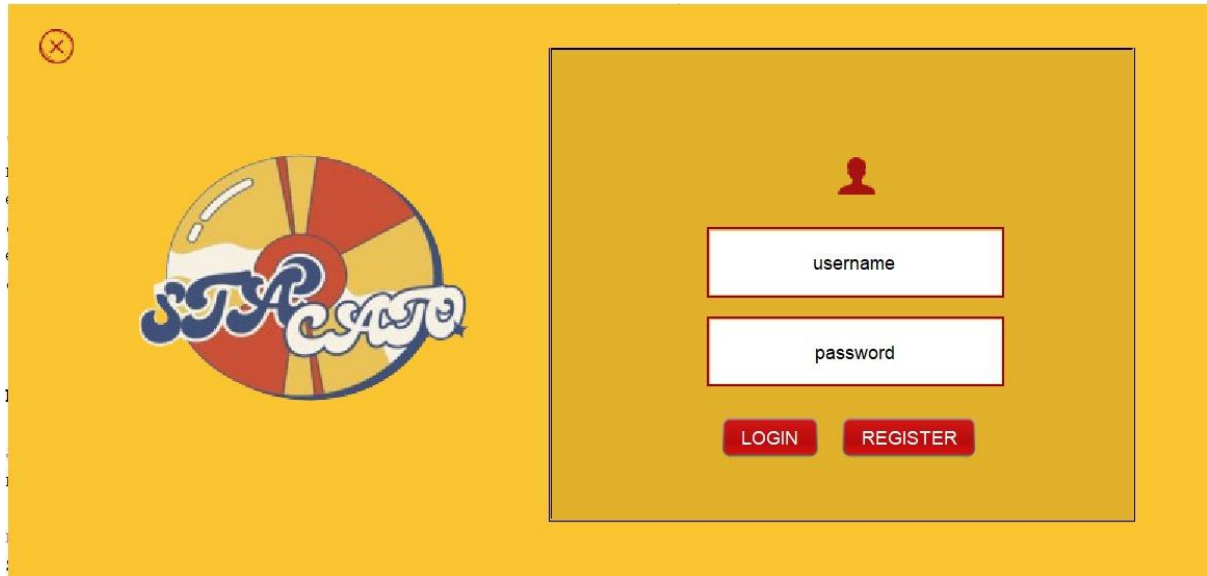
    /**
     * Creates new form Register1
     */
    public Register1() throws SQLException {
        initComponents();
        txID.setEnabled(false);
        autonumber();
    }
}

```

## User Interface Design

### LOGIN

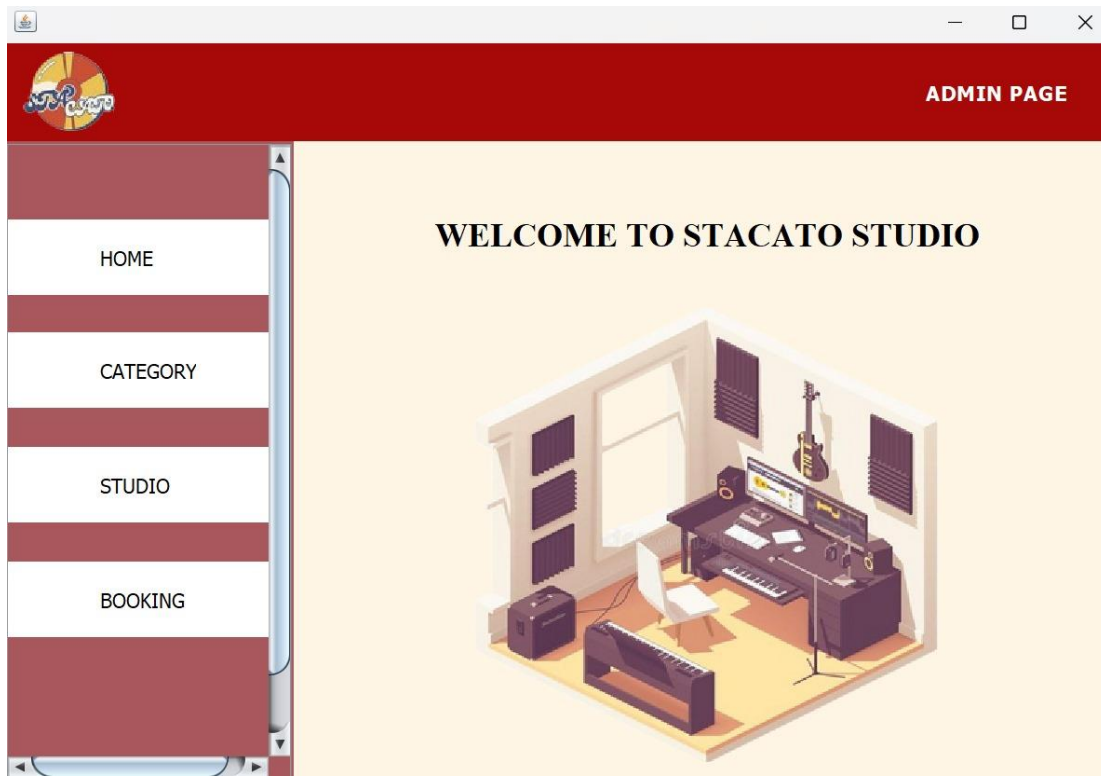
The function of login in a music studio booking application is to allow users to access restricted features or to identify users individually, such as storing preferences, order history, and profile information. It also helps maintain the security of user data and transactions.



The image shows a user interface design for a login page. The background is a solid yellow color. In the top-left corner, there is a small red circle with a white 'X' inside. On the left side, there is a circular logo with a red and yellow striped background and the text 'STARS' in a stylized, blue and white font. On the right side, there is a white rectangular box with a thin black border. Inside this box, at the top, is a small red silhouette of a person. Below this, there are two white rectangular input fields with thin black borders. The first field is labeled 'username' and the second field is labeled 'password'. At the bottom of the box, there are two red rectangular buttons with white text. The first button is labeled 'LOGIN' and the second button is labeled 'REGISTER'.

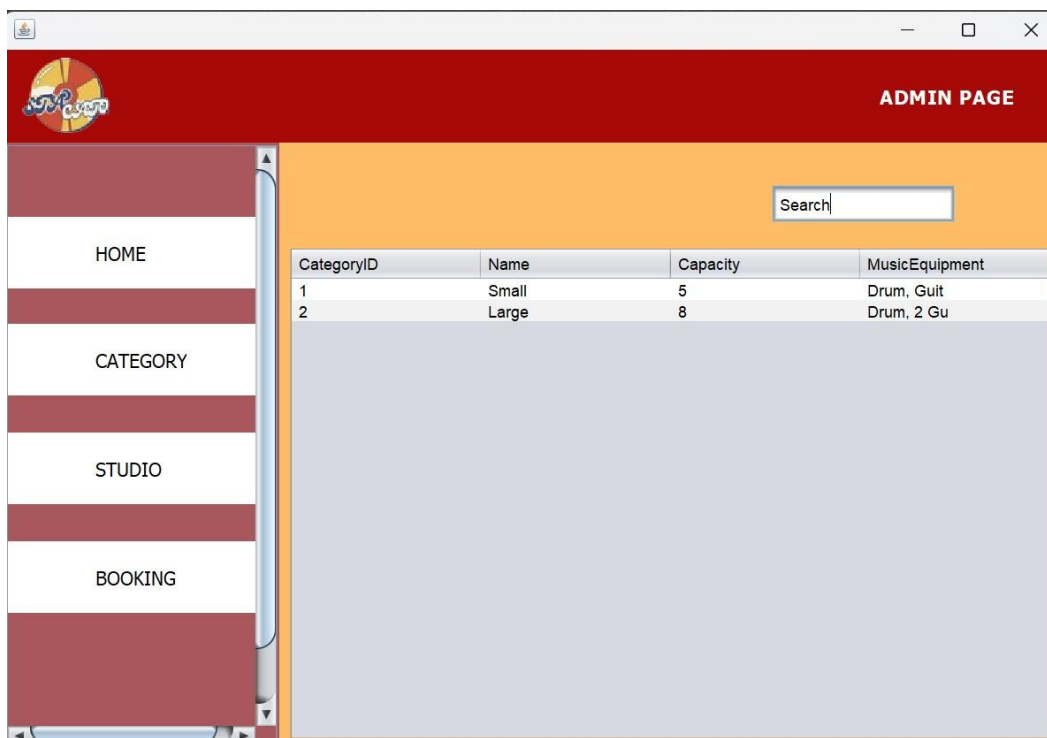
## Home

The home function typically serves as a dashboard where administrators can monitor and manage various aspects of the booking system.



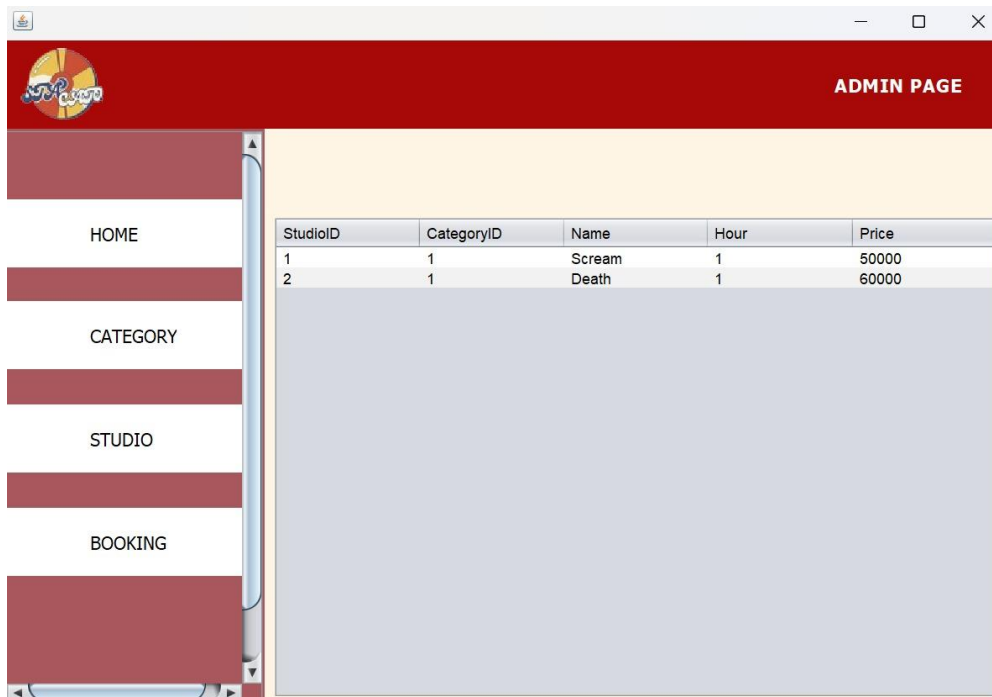
## Category

The category function helps to search for and identify the types of studios available, such as large and small, as shown in the example in the image.



## Studio

The studio function allows you to view the available studio names along with their respective hours and prices.

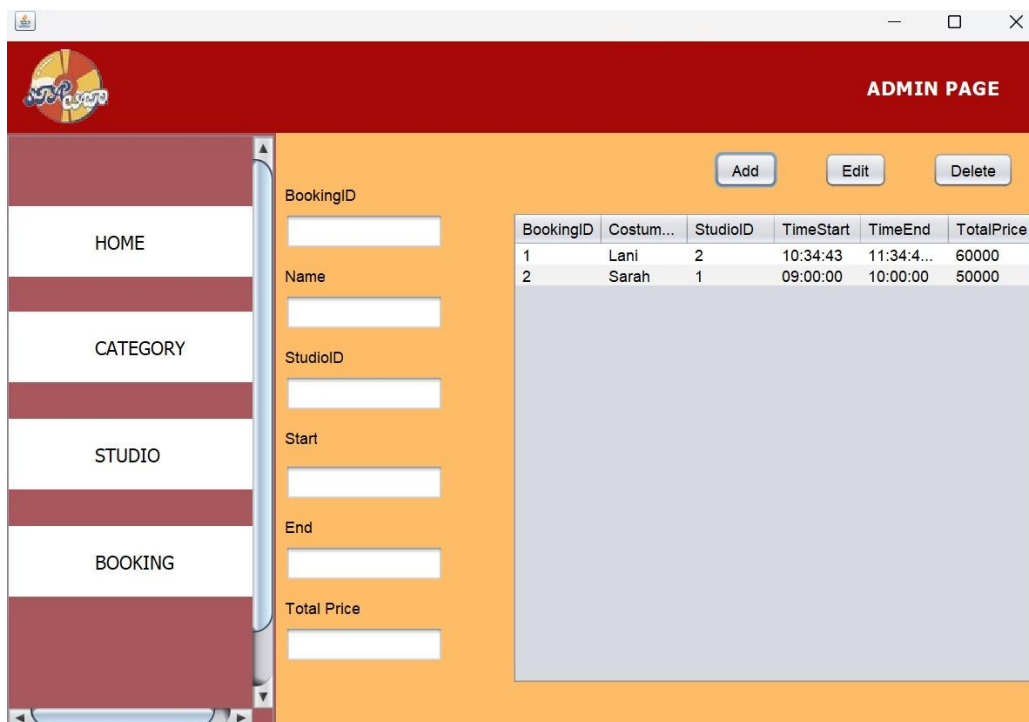


The screenshot shows the 'ADMIN PAGE' for the 'STUDIO' function. The left sidebar contains a menu with 'HOME', 'CATEGORY', 'STUDIO', and 'BOOKING'. The main content area displays a table with the following data:

StudioID	CategoryID	Name	Hour	Price
1	1	Scream	1	50000
2	1	Death	1	60000

## Booking

The booking function is used to monitor, add, edit, and delete users.



The screenshot shows the 'ADMIN PAGE' for the 'BOOKING' function. The left sidebar contains a menu with 'HOME', 'CATEGORY', 'STUDIO', and 'BOOKING'. The main content area features a form for adding, editing, or deleting bookings, and a table displaying the current bookings.

**Form Fields:**

- BookingID:
- Name:
- StudioID:
- Start:
- End:
- Total Price:

**Buttons:** Add, Edit, Delete

**Table:**

BookingID	Costum...	StudioID	TimeStart	TimeEnd	TotalPrice
1	Lani	2	10:34:43	11:34:4...	60000
2	Sarah	1	09:00:00	10:00:00	50000