

PRACTICE REPORT
OBJECT ORIENTED PROGRAMMING LAB WORKS
MODULE
X
PROJECT GUI



Assembled by:

Asy Syifa Najla Khoirunnisa
Farah Faizah
Onic Agustino

L200234269
L200234060
L200234275

INFORMATICS ENGINEERING
FACULTY OF COMMUNICATION AND
INFORMATICS
UNIVERSITAS MUHAMMADIYAH SURAKARTA
2023/2024

We decided to create an application with a graphical interface (GUI) that carries the theme “Hotel Payment System”. This project is designed to fulfill the need for a modern and efficient system in managing various hospitality processes, such as user login, new account registration, room booking, to data management by administrators. By presenting these features, we hope that the system we create can provide convenience and comfort for both customers and hotel managers.

1. Code of class login

- a. This code is used to import the libraries needed in the program

```
package Hotel;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
```

- b. This function is used to set the layout and properties of GUI components, such as labels, buttons, and text fields.

```
public login() {
    initComponents();
}
```

- c. When the “Login” button is pressed, the **btnMoveActionPerformed()** function will be called.

```
private void btnRegActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    new register().setVisible(true);
    this.dispose();
}
```

```
private void btnMoveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String username = txtUser.getText();
    String password = new String(txtPassword.getPassword());
```

- d. If the user does not fill in the username or password, an error message will appear.

```
if (username.isEmpty() || password.isEmpty()) {
    JOptionPane.showMessageDialog(this, "Please fill all fields", "Error", JOptionPane.ERROR_MESSAGE);
} else {
```

- e. Open a connection to the database using the **DatabaseConnection.configDB()** function.

```
try (Connection conn = DatabaseConnection.configDB()) {
```

- f. This query checks if the username and password match in the login table.

```
String sql = "SELECT * FROM login WHERE username = ? AND password = ?";
```

- g. If the **username is admin**, the user will be directed to the admin page (Admin_Page).
 If the **username is normal**, the user is redirected to the hotel's main page (HalUtama).
 If the **username and password are invalid**, an error message is displayed.

```
if (txtUser.getText().equalsIgnoreCase("admin") && rs.next()){
    // Proceed to the next page or functionality
    new Admin_Page().setVisible(true);
    this.dispose();
    JOptionPane.showMessageDialog(null, "Selamat datang " + txtUser.getText() + " di Aplikasi Admin Control Hotel");
} else if(rs.next()){
    new HalUtama().setVisible(true);
    this.dispose();
    JOptionPane.showMessageDialog(null, "Selamat datang " + txtUser.getText() + " di Aplikasi Hotel");
} else {
    JOptionPane.showMessageDialog(this, "Invalid username or password", "Error", JOptionPane.ERROR_MESSAGE);
}
```

- h. **main function** Create and display the login page

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(() -> {
        new login().setVisible(true); // Tampilka
    });
}
```

- i. if there is an error in the connection or **SQL query**, an error message will be displayed to the user.

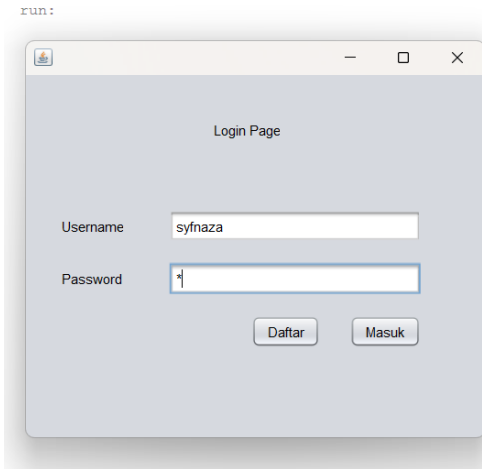
```
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
}
```

- j. These variables represent the elements used in the GUI

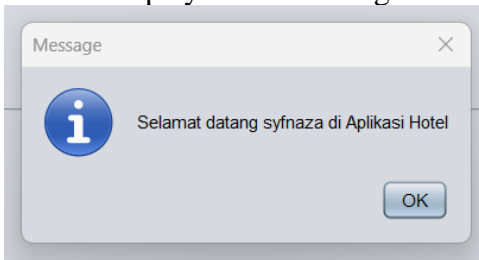
```
// Variables declaration - do not modify
private javax.swing.JButton btnMove;
private javax.swing.JButton btnReg;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JPasswordField txtPassword;
private javax.swing.JTextField txtUser;
// End of variables declaration
```

- k. Enter username and password to enter the booking page, if using a special username and password admin then automatically enter the room control page.

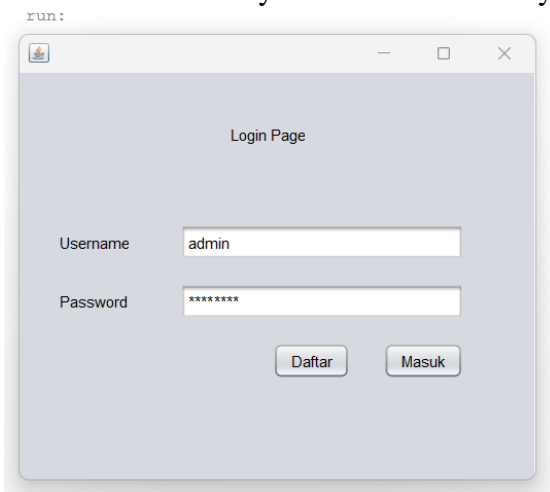
If you have never registered or as a new user then you can press the register button so that it will be redirected to the register page.









- l. Display when entering the booking page



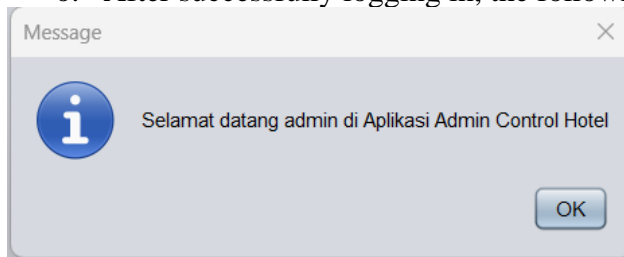
- m. for the admin page we need a special user to enter the admin page like this example we use the user “admin” that has been registered, so that if you use this username it will automatically be redirected directly to the admin page for room control.



n. the display for database

				id	username	password
<input type="checkbox"/>		Edit		Copy		Delete
	1	dara				dara123
<input type="checkbox"/>		Edit		Copy		Delete
	20	admin				admin123

o. After successfully logging in, the following display will appear



2. Code of register class

a. This code is used to import the libraries needed in the program

```
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
import javax.swing.JOptionPane;
```

b. **Public class register:** Defines the register class as a GUI window

```
public class register extends JFrame {
```

c. This function is used to set the layout and properties of GUI components, such as labels, buttons, and text fields.

```
public register() {  
    initComponents();  
}
```

d. enter username and password

```
private void btnRegActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    String username = txtUser1.getText();  
    String password = new String(txtPass.getPassword());
```

e. Ensure both inputs are not empty. If empty, display an error message using **JOptionPane**.

```
if (username.isEmpty() || password.isEmpty()) {  
    JOptionPane.showMessageDialog(this, "Please fill all field", "Error", JOptionPane.ERROR_MESSAGE);
```

f. Open a connection to the database via **DatabaseConnection.configDB()**.

g. Set up the **SQL INSERT** command to add user data to the login table.

h. Runs the **executeUpdate()** command to save the data to the database.

```

try (Connection conn = DatabaseConnection.configDB()) {
    String sql = "INSERT INTO login (username, password) VALUES (?,?)";
    try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, username);
        pstmt.setString(2, password);
        pstmt.executeUpdate();
        JOptionPane.showMessageDialog(this, "User registered successfully", "Success", JOptionPane.INFORMATION_MESSAGE);
    }
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
}

```

- i. Opens the login page by creating an instance of the login class.
- j. Closes the current registration page using **this.dispose()**.

```

private void btnLogActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    new login().setVisible(true);
    this.dispose();
}

```

- k. **invokeLater**: Ensures the GUI is run in the Event Dispatch Thread (EDT) thread to prevent threading issues.
- l. **new register().setVisible(true);**: Creates and displays the registration window.

```

public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new register().setVisible(true);
        }
    });
}

```

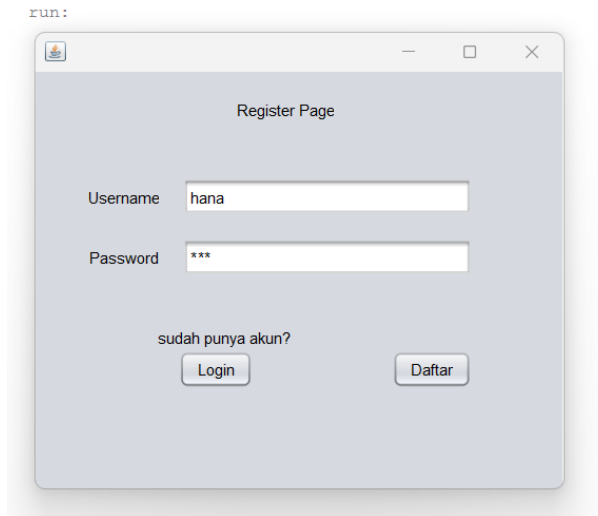
- m. These variables represent the elements used in the GUI

```

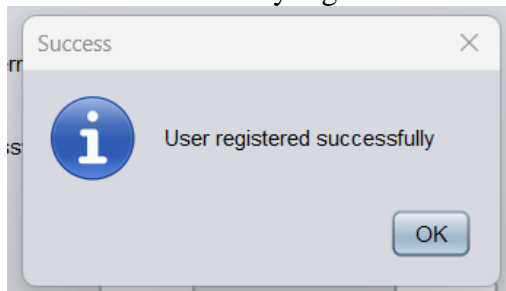
private javax.swing.JButton btnLog;
private javax.swing.JButton btnReg1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JPasswordField txtPass;
private javax.swing.JTextField txtUser1;
// End of variables declaration

```













- n. The register page functions for users who have never logged in, so create a new account to be able to log in then press the register button.



- o. If successfully register then it will appear like this



- p. The result in database

				id	username	password
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	dara	dara123
<input type="checkbox"/>	 Edit	 Copy	 Delete	20	admin	admin123
<input type="checkbox"/>	 Edit	 Copy	 Delete	21	syfnaza	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	23	hana	123

3. Code of admin_page class

- a. This code is used to import the libraries needed in the program

```
import java.awt.HeadlessException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.Connection;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import javax.swing.RowFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
```

- b. **Admin_Page** is a public class extended from javax.swing.JFrame, the main class for creating application windows.
- c. **DefaultTableModel** is the data model used for tables (JTable). This model makes it easy to manipulate table data, such as adding, deleting, or modifying data.

```
public class Admin_Page extends javax.swing.JFrame {
    DefaultTableModel model = new DefaultTableModel();
```

- d. **TableRooms** is the table where the room data is stored.
- e. **TableRowSorter** is a tool that can be used to filter or sort the data in the table.
- f. Once the sorter tool is created, we connect it to the TableRooms table. Now the table can be set to only show certain data.
- g. **RowFilter.regexFilter(str)** is the part that filters the data. If the keyword str is “Deluxe”, the table will only display rooms that have the word “Deluxe”.

```
public void cari(String str) {
    model = (DefaultTableModel) TableRooms.getModel();
    TableRowSorter<DefaultTableModel> trs = new TableRowSorter<>(model);
    TableRooms.setRowSorter(trs);
    trs.setRowFilter(RowFilter.regexFilter(str));
}
```

- h. This method is used for **TableBooking** tables, not TableRooms.

```
public void search(String str) {
    model = (DefaultTableModel) TableBooking.getModel();
    TableRowSorter<DefaultTableModel> trs = new TableRowSorter<>(model);
    TableBooking.setRowSorter(trs);
    trs.setRowFilter(RowFilter.regexFilter(str));
}
```

- i. Create a **DefaultTableModel** object to store the data to be displayed in the table

```
private void tampilkan_dataBooking() {
    DefaultTableModel model = new DefaultTableModel();
```


j. **Add columns** to the table model.

```
model.addColumn("ID");
model.addColumn("NIK");
model.addColumn("Nama");
model.addColumn("Email");
model.addColumn("No Hp");
model.addColumn("ID Kamar");
model.addColumn("Check In");
model.addColumn("Check Out");
model.addColumn("Total Harga");
model.addColumn("Status");
```

k. **Create an SQL** command to retrieve all data from the booking table in the database

l. **createStatement()**: Creates a Statement object to execute the SQL command

m. **executeQuery(sql)**: Executes the SQL query and stores the results in a ResultSet object

```
try{
    String sql = "SELECT * FROM booking";
    java.sql.Connection conn = (Connection)DatabaseConnection.configDB();
    java.sql.Statement stm = conn.createStatement();
    java.sql.ResultSet res = stm.executeQuery(sql);
```

n. **res.next()**: Iterate to read each row of data in the ResultSet. As long as there are rows of data, the loop will continue to run.

```
while(res.next()){
    model.addRow(new Object[]{res.getString(1), res.getString(2), res.getString(3), res.getString(4),
    res.getString(5), res.getString(6), res.getString(7), res.getString(8), res.getString(9), res.getString(10)});
}
```

o. Connects the populated data model to the **TableBooking** table component.

```
TableBooking.setModel(model);
```

p. **catch (SQLException ex)**: Catches an error if a problem occurs while retrieving data from the database

```
}catch(SQLException ex){
    Logger.getLogger(Admin_Page.class.getName()).log(Level.SEVERE, null, ex);
}
```

q. Same process as the `tampilkan_dataBooking` method

```
private void tampilkan_data() {
    DefaultTableModel model = new DefaultTableModel();
    model.addColumn("ID");
    model.addColumn("No Kamar");
    model.addColumn("Type Kamar");
    model.addColumn("Fasilitas");
    model.addColumn("Harga");
    model.addColumn("Status");

    try {
        String sql = "SELECT * FROM rooms";
        java.sql.Connection conn = (Connection) DatabaseConnection.configDB();
        java.sql.Statement stm = conn.createStatement();
        java.sql.ResultSet res = stm.executeQuery(sql);

        while (res.next()) {
            model.addRow(new Object[] { res.getString(1), res.getString(2), res.getString(3), res.getString(4),
            res.getString(5), res.getString(6) });
        }
        TableRooms.setModel(model);
    } catch (SQLException ex) {
        Logger.getLogger(Admin_Page.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

r. `sql` is an **SQL string** that is used to **add** new data to the rooms table in the database.

s. After the data is successfully added to the database, **`JOptionPane.showMessageDialog()`** will display a success message to the user.

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        String sql = "INSERT INTO rooms VALUES ('"+txtIdKamar.getText()+"', '"+txtNoKamar.getText()+"', '"+cmbType.getSelectedItem().toString()+"', '"+txtFasilitas.getText()+"', '"+txtPrice.getText()+"', '"+cmbStatus.getSelectedItem().toString()+"')";

        java.sql.Connection conn = (Connection) DatabaseConnection.configDB();
        java.sql.PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.execute();
        JOptionPane.showMessageDialog(null, "Proses Add Kamar Berhasil");
        tampilkan_data();
        kosongkan_form();
    } catch (SQLException ex) {
        Logger.getLogger(Admin_Page.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

t. `sql` is an **SQL string** that is used to **delete** data to the rooms table in the database

```
private void btnDelActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        String sql = "DELETE FROM rooms WHERE id_room='"+txtIdKamar.getText()+"'";

        java.sql.Connection conn = (Connection) DatabaseConnection.configDB();
        java.sql.PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.execute();
        JOptionPane.showMessageDialog(null, "Hapus Data Rooms Berhasil");
        kosongkan_form();
    } catch (HeadlessException | SQLException e) {
        System.out.println("Error : " + e.getMessage());
    }

    tampilkan_data();
    kosongkan_form();
}
```

u. sql is an **SQL string** that is used to **update** data to the rooms table in the database

```
private void btnEditActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try{
        String sql = "UPDATE rooms SET id_room = '"+txtIdKamar.getText()+"', no_room='"+txtNoKamar.getText()+
            "', type_room='"+cmbType.getSelectedItem().toString()+"'"+
            ", fasilitas='"+txtFasilitas.getText()+"', price='"+txtPrice.getText()+
            "', status='"+cmbStatus.getSelectedItem().toString()+"' WHERE id_room = '"+txtIdKamar.getText()+"'";

        java.sql.Connection conn = (Connection)DatabaseConnection.configDB();
        java.sql.PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.execute();
        JOptionPane.showMessageDialog(null, "Edit Data Rooms Berhasil");
    } catch (HeadlessException | SQLException e) {
        System.out.println("Error : " + e.getMessage());
    }
    tampilkan_data();
    kosongkan_form();
}
```

v. **vt.getPoint()**: Retrieves the mouse position when clicking on the TableRooms table

```
private void TableRoomsMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int tabel = TableRooms.rowAtPoint(evt.getPoint());
```

w. Retrieve Data from a Table Based on Clicked Rows

```
String id_room = TableRooms.getValueAt(tabel, 0).toString();
txtIdKamar.setText(id_room);

String no_room = TableRooms.getValueAt(tabel, 1).toString();
txtNoKamar.setText(no_room);

String type_room = TableRooms.getValueAt(tabel, 2).toString();
cmbType.setSelectedItem(type_room);

String fasilitas = TableRooms.getValueAt(tabel, 3).toString();
txtFasilitas.setText(fasilitas);

String price = TableRooms.getValueAt(tabel, 4).toString();
txtPrice.setText(price);

String status = TableRooms.getValueAt(tabel, 5).toString();
cmbStatus.setSelectedItem(status);
```

- x. **txtSearchActionPerformed**: used to filter or search for data in a table based on the entered text.
- y. The **txtcari** and **txtsearch** functions are similar, but may be used for different tables in the application.
- z. **btnBackActionPerformed**: This method is executed when the “Back” button is pressed by the user.

```
private void txtCariActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String searchString = txtCari.getText();
    cari(searchString);
}

```

```
private void txtSearchActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String searchString1 = txtSearch.getText();
    search(searchString1);
}

```

```
private void btnBackActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    new login().setVisible(true);
    this.dispose();
}

```

- aa. Opens and displays the frame of the **Admin_Page** class Using **invokeLater()** to ensure the UI is executed in the event dispatch thread

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Admin_Page().setVisible(true);
        }
    });
}

```

- bb. Display for control room that can add, edit, and delete the data

Rooms Control		
ID	<input type="text" value="7"/>	<input type="button" value="Add"/>
No Kamar	<input type="text" value="C34"/>	<input type="button" value="Edit"/>
Type Kamar	<input type="text" value="Single Room"/>	<input type="button" value="Delete"/>
Fasilitas	<input type="text" value="Sarapa pagi"/>	
Harga	<input type="text" value="2000000"/>	
Status	<input type="text" value="Available"/>	

cc. Here is a table to display available rooms, there is also a button to return to the login page, then there is a field to search for the room you want to search for.

						Back
						<input type="text"/>
ID	No Kamar	Type Kamar	Fasilitas	Harga	Status	
1	B12	Deluxe Room	Tv	1000000	available	
2	V12	Double Room	Lengkap	1290000	maintenance	
3	A13	Standard Room	AC	100000	available	
4	H17	Twin Room	Wifi	3000000	maintenance	
5	G67	Suite Room	Kolam	900000	booked	
6	U56	Single Room	Gym	4999999	available	
7	C34	Family Room	Sarapan pagi	2000000	available	

dd. Display in database

		id_room	no_room	type_room	fasilitas	price	status
<input type="checkbox"/>	Edit Copy Delete	1	B12	Deluxe Room	Tv	1000000	available
<input type="checkbox"/>	Edit Copy Delete	2	V12	Double Room	Lengkap	1290000	maintenance
<input type="checkbox"/>	Edit Copy Delete	3	A13	Standard Room	AC	100000	available
<input type="checkbox"/>	Edit Copy Delete	4	H17	Twin Room	Wifi	3000000	maintenance
<input type="checkbox"/>	Edit Copy Delete	5	G67	Suite Room	Kolam	900000	booked
<input type="checkbox"/>	Edit Copy Delete	6	U56	Single Room	Gym	4999999	available
<input type="checkbox"/>	Edit Copy Delete	7	C34	Family Room	Sarapan pagi	2000000	available

ee. In this section there is a history of room bookings

										<input type="text"/>
ID	NIK	Nama	Email	No Hp	ID Kamar	Check In	Check Out	Total Harga	Status	
1	7868333	Onic	onic@gmail.com	08756725782	2	2024-12-31	2025-01-01	1290000.0	booked	
2	897987	Heru	heru@gmail.com	075355367	1	2025-01-01	2025-01-24	2.3E7	booked	
3	9356536	Najla	najla@gmail.com	085653633	6	2024-12-31	2025-01-08	3.99999992E7	booked	
4	4736733	Farah	farah@gmail.com	0845378783	3	2024-12-31	2025-01-04	400000.0	booked	
5	12345	syfnaza	syfnaza@gmail.com	087653462133	1	2025-01-03	2025-01-04	1000000.0	booked	
6	89064	hasna	hasna@gmail.com	87654321906	5	2025-01-08	2025-01-10	1800000.0	booked	
7	12	na	na@gmail.com	123	5	2025-01-08	2025-01-10	1800000.0	booked	

4. Code of HallUtama class

- public class HalUtama:** The main class that is derived from **JFrame**. Used to create the main window of the application.

- DefaultTableModel model:** An object used to manage data in a table.

```
public class HalUtama extends javax.swing.JFrame {
    DefaultTableModel model = new DefaultTableModel();
```

- txtIn:** The JDateChooser component used to select the check-in date.
- calculateTotalPrice():** This method will calculate the total price based on the check-in

date change.

- e. **txtIdRoom**: A JTextField component for entering the room ID.

```
private void addEventListeners() {
    // Listener untuk JDateChooser Check In
    txtIn.addPropertyChangeListener("date", evt -> hitungTotalHarga());

    // Listener untuk JDateChooser Check Out
    txtOut.addPropertyChangeListener("date", evt -> hitungTotalHarga());

    // Listener untuk TextField ID Kamar
    txtIdKamar.addActionListener(evt -> hitungTotalHarga());
}
```

- f. Create a method that checks whether all the inputs needed to calculate the total price have been filled in.

```
private void hitungTotalHarga() {
    try {
        // Pastikan semua input yang dibutuhkan sudah diisi
        if (txtIn.getDate() == null || txtOut.getDate() == null || txtIdKamar.getText().isEmpty()) {
            txtTotal.setText("0");
            return;
        }
    }
}
```

- g. To calculate the length of stay

```
java.util.Date checkIn = txtIn.getDate();
java.util.Date checkOut = txtOut.getDate();
long diffInMillies = Math.abs(checkOut.getTime() - checkIn.getTime());
long days = diffInMillies / (1000 * 60 * 60 * 24);
```

- h. Retrieve the room price (price) from the rooms table based on id_room.

```
String sql = "SELECT price FROM rooms WHERE id_room = ?";
Connection conn = DatabaseConnection.configDB();
java.sql.PreparedStatement pst = conn.prepareStatement(sql);
pst.setString(1, txtIdKamar.getText());
java.sql.ResultSet rs = pst.executeQuery();
```

- i. **txtTotal.setText(...)**: Displays the total price calculated in the txtTotal column

```
if (rs.next()) {
    double hargaKamar = rs.getDouble("price");
    double totalHarga = days * hargaKamar;
    txtTotal.setText(String.valueOf(totalHarga));
}
```

- j. Display a warning and error message using **JOptionPane** if the room ID is not found

```
} else {
    txtTotal.setText("0");
    JOptionPane.showMessageDialog(this, "ID Kamar tidak ditemukan.", "Peringatan", JOptionPane.WARNING_MESSAGE);
}
} catch (SQLException ex) {
    Logger.getLogger(HalUtama.class.getName()).log(Level.SEVERE, null, ex);
    JOptionPane.showMessageDialog(this, "Terjadi kesalahan saat menghitung total harga: " + ex.getMessage());
}
```

k. Create a table model to display data from database to table (TableRooms)

```
private void tampilkan_dataKamar() {  
    DefaultTableModel model = new DefaultTableModel();  
    model.addColumn("ID");  
    model.addColumn("No Kamar");  
    model.addColumn("Type Kamar");  
    model.addColumn("Fasilitas");  
    model.addColumn("Harga");  
    model.addColumn("Status");  
}
```

l. **res.next()**: Retrieves the next row from the query result. Stops if there are no more rows.

```
while(res.next()) {  
    model.addRow(new Object[]{res.getString(1), res.getString(2), res.getString(3),  
    res.getString(4), res.getString(5), res.getString(6)});  
}
```

m. Set the TableRooms table model using the data that has been inserted into the DefaultTableModel.

```
TableRooms.setModel(model);
```

n. **model.addRow(...)**: Adds a row of data to the table model.

```
model.addRow(new Object[]{  
    res.getString("id_booking"),  
    res.getString("nik"),  
    res.getString("nama"),  
    res.getString("email"),  
    res.getString("no_hp"),  
    res.getString("id_room"),  
    formattedCheckIn,  
    formattedCheckOut,  
    res.getString("total_price"),  
    res.getString("status_booking")  
});
```

o. If the booking data is only one row (the first row), then the data from that row is displayed directly to some input components in the GUI.

```
if (model.getRowCount() == 1) {  
    java.util.Date utilCheckIn = sqlCheckIn != null ? new java.util.Date(sqlCheckIn.getTime()) : null;  
    java.util.Date utilCheckOut = sqlCheckOut != null ? new java.util.Date(sqlCheckOut.getTime()) : null;  
  
    txtIn.setDate(utilCheckIn);  
    txtOut.setDate(utilCheckOut);  
  
    txtNIK.setText(res.getString("nik"));  
    txtNama.setText(res.getString("nama"));  
    txtEmail.setText(res.getString("email"));  
    txtNoHp.setText(res.getString("no_hp"));  
    txtIdKamar.setText(res.getString("id_room"));  
    txtTotal.setText(res.getString("total_price"));  
    cmbStatus.setSelectedItem(res.getString("status_booking")); // Set status kamar  
}
```

- p. Set the value of the text component (**JTextField**) to null.

```
private void kosongkan form() {  
    txtIdBooking.setText(null);  
    txtNIK.setText(null);  
    txtNama.setText(null);  
    txtEmail.setText(null);  
    txtNoHp.setText(null);  
    txtIdKamar.setText(null);  
    txtIn.setDate(null);  
    txtOut.setDate(null);  
    txtTotal.setText(null);  
    cmbStatus.setSelectedItem(0);  
};
```

- q. Converts dates in **java.util.Date** format to **java.sql.Date** for use in SQL queries.

```
private boolean isKamarAvailable(String idRoom, java.util.Date checkIn, java.util.Date checkOut) {  
    try {  
        // Konversi tanggal ke tipe java.sql.Date  
        java.sql.Date sqlCheckIn = new java.sql.Date(checkIn.getTime());  
        java.sql.Date sqlCheckOut = new java.sql.Date(checkOut.getTime());
```

- r. This SQL used to calculate the number of booking conflicts works so that new dates should not overlap with existing bookings for that room.

```
String sql = "SELECT COUNT(*) AS total FROM booking " +  
            "WHERE id_room = ? " +  
            "AND (check_in <= ? AND check_out >= ?)";
```

- s. Executes the SQL query and gets the result in the form of **ResultSet**.

```
java.sql.ResultSet rs = pstmt.executeQuery();  
  
if (rs.next()) {  
    int totalConflicts = rs.getInt("total");  
    // Jika ada konflik (total > 0), kamar tidak tersedia  
    return totalConflicts == 0;  
}
```

- t. If an error occurs or no data is found, the method will return false as the default value, indicating that the room is not available.

```
return false; //
```

- u. The **public HalUtama()** method is the constructor of the HalUtama class

```
public HalUtama() {  
    initComponents();  
    tampilkan_data();  
    kosongkan_form();  
    tampilkan_dataKamar();  
    hitungTotalHarga();  
    addEventListeners();  
}
```


- v. `utilCheckIn` and `utilCheckOut`: Retrieve the Check-In and Check-Out dates from the `JDateChooser` component.

```
private void btnPesanActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
        // Ambil nilai tanggal dari JDateChooser  
        java.util.Date utilCheckIn = txtIn.getDate();  
        java.util.Date utilCheckOut = txtOut.getDate();
```

- w. Ensures the user has selected the Check-In and Check-Out dates. If either date is empty, the app displays a warning message and the process is terminated.

```
if (utilCheckIn == null || utilCheckOut == null) {  
    JOptionPane.showMessageDialog(this, "Harap pilih tanggal Check In dan Check Out.", "Peringatan", JOptionPane.WARNING_MESSAGE);  
    return;  
}
```

- x. Check if the room is available on the selected date by calling the `isRoomAvailable` method.

```
if (!isKamarAvailable(txtIdKamar.getText(), utilCheckIn, utilCheckOut)) {  
    JOptionPane.showMessageDialog(this, "Kamar sudah dipesan pada tanggal tersebut!", "Error", JOptionPane.ERROR_MESSAGE);  
    return;  
}
```

- y. Establish a connection to the database and prepare an SQL query to add booking data to the booking table.

```
String sql = "INSERT INTO booking (id_booking, nik, nama, email, no_hp, id_room, check_in, check_out, total_price,"  
            + " status_booking) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";  
Connection conn = DatabaseConnection.configDB();  
java.sql.PreparedStatement pstmt = conn.prepareStatement(sql);
```

- z. To Fill the SQL query parameters with data from the form.

```
pstmt.setString(1, txtIdBooking.getText());  
pstmt.setString(2, txtNIK.getText());  
pstmt.setString(3, txtNama.getText());  
pstmt.setString(4, txtEmail.getText());  
pstmt.setString(5, txtNoHp.getText());  
pstmt.setString(6, txtIdKamar.getText());  
pstmt.setDate(7, sqlCheckIn);  
pstmt.setDate(8, sqlCheckOut);  
pstmt.setString(9, txtTotal.getText());  
pstmt.setString(10, cmbStatus.getSelectedItem().toString());
```

- aa. Run an `INSERT` query to the database to add booking data.

```
pstmt.executeUpdate();
```

- bb. Notifies the user that the booking process was successful.

```
JOptionPane.showMessageDialog(null, "Proses Pemesanan Berhasil");
```

- cc. `show_data()`: Reloads the booking data from the database to update the table.

- dd. `clear_form()`: Clears the input form so that it is ready for reuse

```
tampilkan_data();  
kosongkan_form();
```

ee. This method handles the event when the user clicks on the **TableBooking** table to select a row.

```
private void TableBookingMouseClicked(java.awt.event.MouseEvent evt) {  
    // TODO add your handling code here:  
    int tabel = TableBooking.rowAtPoint(evt.getPoint());  
  
    String id_booking = TableBooking.getValueAt(tabel, 0).toString();  
    txtIdBooking.setText(id_booking);  
  
    String nik = TableBooking.getValueAt(tabel, 1).toString();  
    txtNIK.setText(nik);  
  
    String nama = TableBooking.getValueAt(tabel, 2).toString();  
    txtNama.setText(nama);  
  
    String email = TableBooking.getValueAt(tabel, 3).toString();  
    txtEmail.setText(email);  
  
    String no_hp = TableBooking.getValueAt(tabel, 4).toString();  
    txtNoHp.setText(no_hp);  
  
    String id_room = TableBooking.getValueAt(tabel, 5).toString();  
    txtIdKamar.setText(id_room);  
}
```

ff. Creates a delete query, then executes the query

gg. **JOptionPane()** Notifies the user that the data was successfully deleted.

```
private void btnDelActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        String sql = "DELETE FROM booking WHERE id_booking='"+txtIdBooking.getText()+"'";  
  
        java.sql.Connection conn = (Connection)DatabaseConnection.configDB();  
        java.sql.PreparedStatement pstmt = conn.prepareStatement(sql);  
        pstmt.execute();  
        JOptionPane.showMessageDialog(null, "Hapus Data Booking Berhasil");  
        kosongkan_form();  
    } catch (HeadlessException | SQLException e) {  
        System.out.println("Error : " + e.getMessage());  
    }  
}
```

hh. Ensure that all important fields (check-in, check-out date, room ID, total price) are filled in before proceeding with the update process.

```
private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try {  
        // Validasi input  
        if (txtIn.getDate() == null || txtOut.getDate() == null || txtIdKamar.getText().isEmpty() || txtTotal.getText().isEmpty()) {  
            JOptionPane.showMessageDialog(this, "Harap lengkapi semua data sebelum mengupdate.", "Peringatan",  
                JOptionPane.WARNING_MESSAGE);  
            return;  
        }  
    }  
}
```

ii. `txtIn.getDate()`: Retrieves the date from the `JDateChooser` component.

jj. Convert to `java.sql.Date`: Required to save the date to the database (SQL requires this format).

```
java.util.Date utilCheckIn = txtIn.getDate();
java.util.Date utilCheckOut = txtOut.getDate();
```

```
// Konversi ke java.sql.Date untuk database
java.sql.Date checkIn = new java.sql.Date(utilCheckIn.getTime());
java.sql.Date checkOut = new java.sql.Date(utilCheckOut.getTime());
```

kk. `calculateTotalPrice()`: Recalculates the total price based on the new check-in and check-out dates.

```
hitungTotalHarga();
String totalHarga = txtTotal.getText();
```

ll. UPDATE booking: SQL query to update the data in the booking table.

mm. WHERE `id_booking = ?` A condition to ensure that only data with a specific `id_booking` is updated.

```
String sql = "UPDATE booking SET nik = ?, nama = ?, email = ?, no_hp = ?, id_room = ?, "
            + "check_in = ?, check_out = ?, total_price = ? WHERE id_booking = ?";
Connection conn = DatabaseConnection.configDB();
java.sql.PreparedStatement pst = conn.prepareStatement(sql);
```

nn. Main method in Java, which is used to run Swing-based GUI applications

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new HalUtama().setVisible(true);
        }
    });
}
```

- oo. First page view of booking room contains textfield to fill in booking data, there is a booking history table that has been booked.
- pp. there are message, update, and delete buttons

Pemesanan

Cek Kamar

Form Pemesanan

Cari

ID Booking

7

NIK

1233

Nama

hana

Email

hana@gmail.com

No Hp

12345678

ID Kamar

2

Check In

3 Jan 2025

Check Out

4 Jan 2025

Total Harga

1290000.0

Status

Booked

Pesan

ID	NIK	Nama	Email	No Hp	ID Kamar	Check In	Check Out	Total Harga	Status
1	7868333	Onic	onic@gmail.c...	08756725782	2	2024-12-31	2025-01-01	1290000.0	booked
2	897987	Heru	heru@gamil...	075355367	1	2025-01-01	2025-01-24	2.3E7	booked
3	9356536	Najla	najla@gmail...	085653633	6	2024-12-31	2025-01-08	3.9999992E7	booked
4	4736733	Farah	farah@gmail...	0845378783	3	2024-12-31	2025-01-04	400000.0	booked
5	12345	syfnaza	syfnaza@gm...	087653462133	1	2025-01-03	2025-01-04	1000000.0	booked
6	89064	hasna	hasna@gmai...	87654321906	5	2025-01-08	2025-01-10	1800000.0	booked

Update

Delete

Back

- qq. Display when the process is successful






















Message

i

Proses Pemesanan Berhasil

OK

- rr. Display in database

<div><div>←</div><div>→</div></div>				id_booking	nik	nama	email	no_hp	id_room	check_in	check_out	total_price	status_booking
<input type="checkbox"/>				1	7868333	Onic	onic@gmail.com	08756725782	2	2024-12-31	2025-01-01	1290000.0	booked
<input type="checkbox"/>				2	897987	Heru	heru@gamil.com	075355367	1	2025-01-01	2025-01-24	2.3E7	booked
<input type="checkbox"/>				3	9356536	Najla	najla@gmail.com	085653633	6	2024-12-31	2025-01-08	3.9999992E7	booked
<input type="checkbox"/>				4	4736733	Farah	farah@gmail.com	0845378783	3	2024-12-31	2025-01-04	400000.0	booked
<input type="checkbox"/>				5	12345	syfnaza	syfnaza@gmail.com	087653462133	1	2025-01-03	2025-01-04	1000000.0	booked
<input type="checkbox"/>				6	89064	hasna	hasna@gmail.com	87654321906	5	2025-01-08	2025-01-10	1800000.0	booked
<input type="checkbox"/>				7	1233	hana	hana@gmail.com	12345678	2	2025-01-03	2025-01-04	1290000.0	booked

ss. Table view for available rooms, So for the admin page, the admin user can control the desired rooms, can add or modify the types of rooms that will be available at the hotel.

Pemesanan

Cek Kamar

Cari

ID	No Kamar	Type Kamar	Fasilitas	Harga	Status
1	B12	Deluxe Room	Tv	1000000	available
2	V12	Double Room	Lengkap	1290000	maintenance
3	A13	Standard Room	AC	100000	available
4	H17	Twin Room	Wifi	3000000	maintenance
5	G67	Suite Room	Kolam	900000	booked
6	U56	Single Room	Gym	4999999	available
7	C34	Family Room	Sarapan pagi	2000000	available