

**LAPORAN PRAKTIKUM 2**  
**PEMROGRAMAN BERBASIS OBJEK**



**Disusun Oleh :**  
**SASKIA ALIFAH (2411531002)**

**Dosen Pengampu : Nurfiah, S.ST, M.KOM**

**Departemen Informatika**  
**Fakultas Teknologi Informasi**  
**Universitas Andalas**  
**Tahun 2025**

## *LaundryApps*

### A. TUJUAN PRAKTIKUM

1. Memahami cara mengintegrasikan Java dengan MySQL Java Database Connectivity menggunakan MySQL Connector.
2. Menerapkan operasi CRUD (Create, Read, Update, Delete)
3. Membangun aplikasi sederhana berbasis database yang dapat mengelola data pengguna (user) dengan tampilan (UI), logika program (DAO), dan database.

### B. PENDAHULUAN

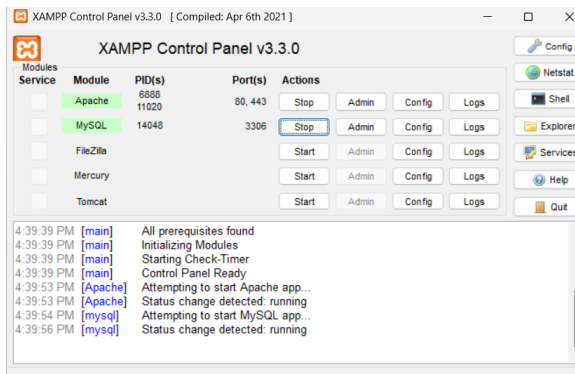
Dalam pembuatan aplikasi berbasis Java, penggunaan database menjadi hal penting untuk menyimpan serta mengelola data agar lebih terstruktur. Java sendiri dapat diintegrasikan dengan berbagai jenis database melalui JDBC (Java Database Connectivity), salah satunya dengan MySQL. Agar aplikasi Java dapat terhubung dengan MySQL, diperlukan driver tambahan berupa MySQL Connector yang berfungsi sebagai jembatan komunikasi antara aplikasi dan database. Dengan koneksi tersebut, program mampu melakukan operasi dasar pengolahan data seperti menambah, menampilkan, memperbarui, dan menghapus data (CRUD).

Database yang digunakan pada praktikum ini adalah MySQL yang dijalankan melalui XAMPP dengan bantuan phpMyAdmin sebagai interface untuk memudahkan pembuatan database dan tabel. Pada contoh ini, dibuat database dengan nama **laundry\_apps** dan tabel **user** yang memiliki beberapa field seperti *id*, *name*, *username*, dan *password*. Tabel inilah yang akan menjadi media penyimpanan data user dan selanjutnya dihubungkan ke aplikasi Java. Dengan begitu, mahasiswa dapat memahami proses kerja sistem penyimpanan data dan cara menghubungkannya ke dalam sebuah aplikasi.

Selain itu, untuk menampilkan data dari database ke dalam aplikasi, digunakan komponen **JTable** yang dipadukan dengan **AbstractTableModel** sehingga data dapat ditampilkan secara rapi dalam bentuk tabel pada GUI. Melalui latihan ini, mahasiswa tidak hanya mempelajari cara membuat koneksi database menggunakan class **Database** dan mengujinya dengan **TestKoneksi**, tetapi juga memahami konsep implementasi **CRUD** dalam aplikasi berbasis Java Swing

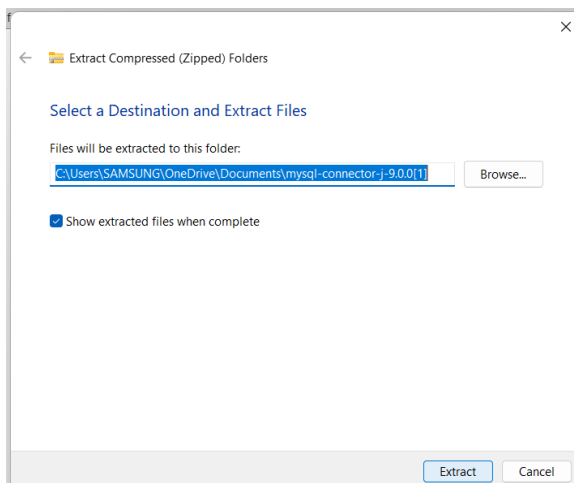
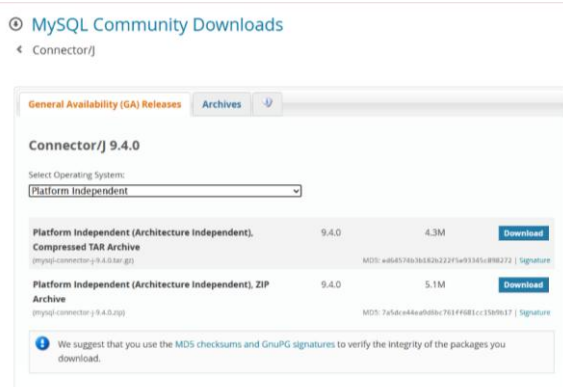
## C. METODE PRAKTIKUM

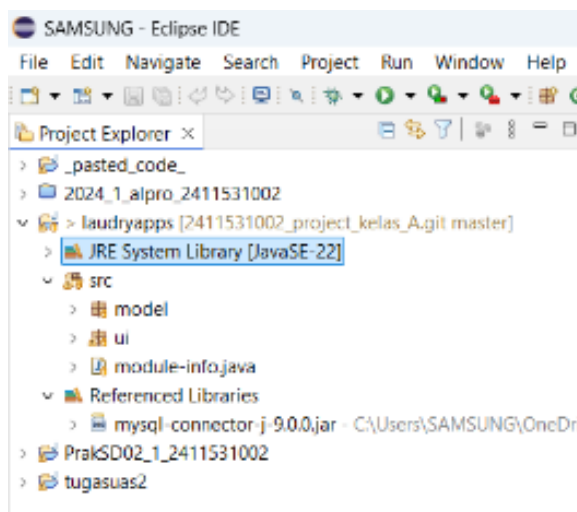
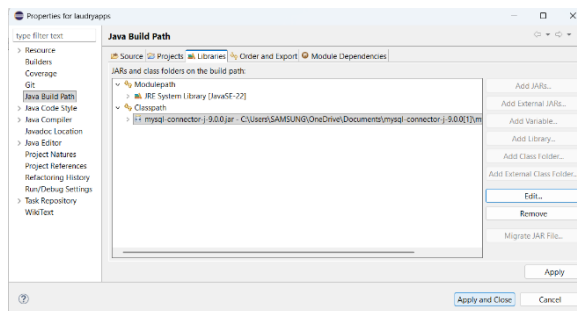
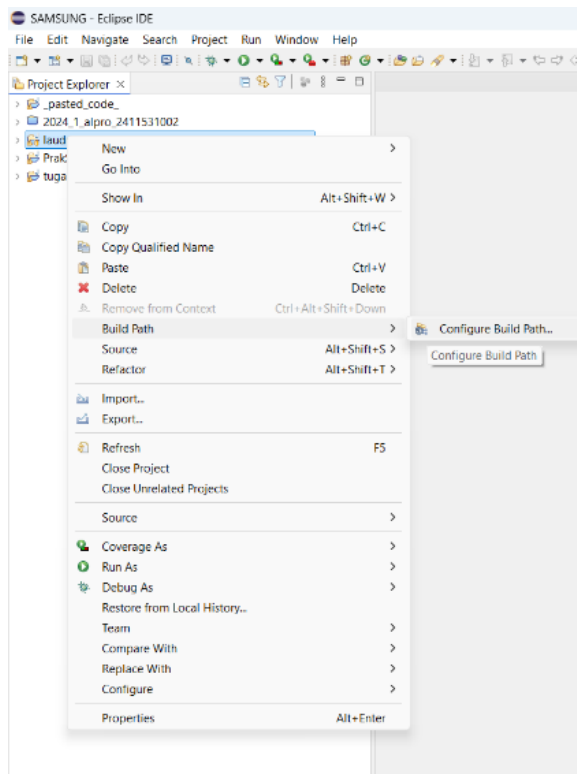
### 1. Install XAMPP



Untuk menggunakan database MySQL, langkah pertama adalah mengunduh XAMPP melalui situs resmi <https://www.apachefriends.org/>. Setelah proses unduhan selesai, lakukan instalasi XAMPP pada komputer atau laptop masing-masing. Selanjutnya, jalankan aplikasi XAMPP dan aktifkan modul **Apache** serta **MySQL**. Dengan begitu, server lokal dan database sudah siap digunakan untuk keperluan pengembangan aplikasi.

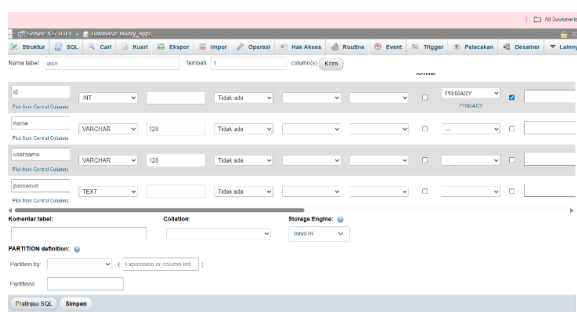
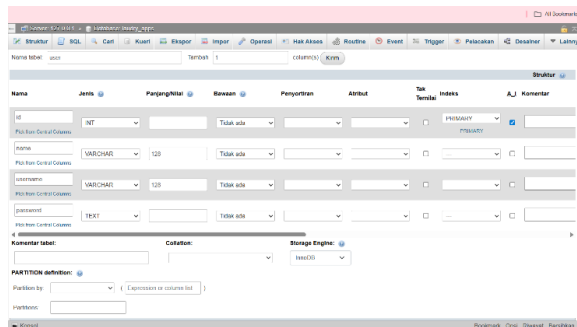
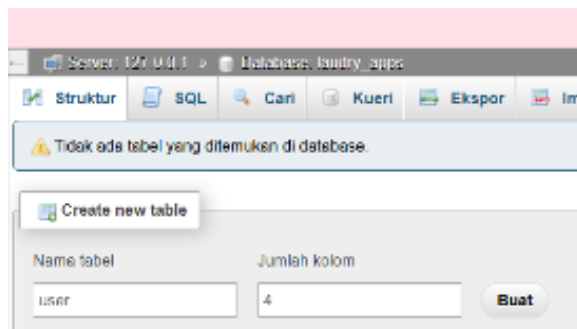
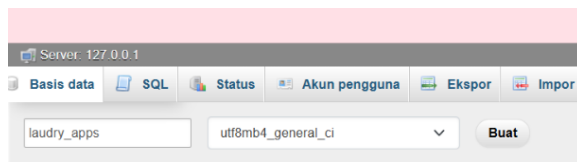
### 2. Menambahkan MySQL Connector





Agar aplikasi Java dapat terhubung dengan MySQL, perlu ditambahkan **MySQL Connector** sebagai driver. Caranya yaitu klik kanan pada JRE System Library → Build Path → Configure Build Path, lalu pilih tab **Libraries** → **Classpath**. Setelah itu klik **Add External JARs**, masukkan file mysql-connector-java yang sudah diunduh, kemudian **Apply and Close**. Jika berhasil, project akan menampilkan folder **Referenced Libraries** yang berisi MySQL Connector sebagai tanda koneksi siap digunakan.

### 3. Membuat Database dan Table User



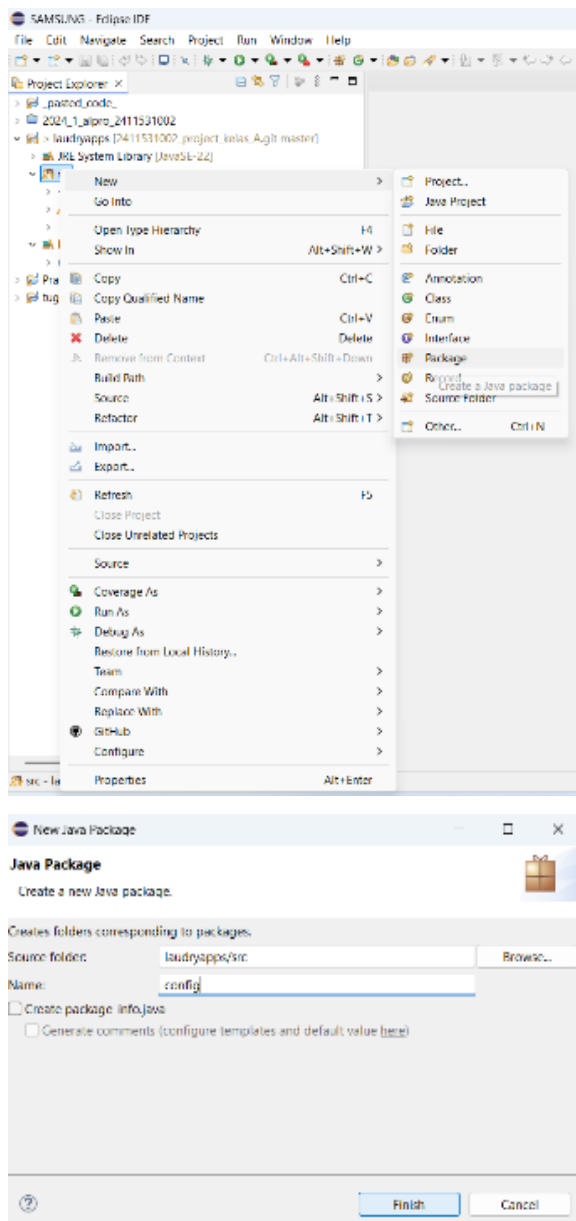
#	Nama	Jenis	Penyortiran	Atribut	Tak Terlihat	Bawaan	Komentar	Ekstra	Tindakan
1	id	INT(11)		Tidak	Tidak ada			AUTO_INCREMENT	Ubah Hapus Lainnya
2	name	varchar(128)		Tidak	Tidak ada				Ubah Hapus Lainnya
3	username	varchar(128)		Tidak	Tidak ada				Ubah Hapus Lainnya
4	password	text		Tidak	Tidak ada				Ubah Hapus Lainnya

Pada tahap ini dilakukan pembuatan database dan tabel user menggunakan phpMyAdmin.

- ✚ Pertama, buka browser dan ketikkan alamat <http://localhost/phpmyadmin>. Ini adalah tampilan antarmuka dari MySQL yang disediakan oleh XAMPP untuk mempermudah pengelolaan database.
- ✚ Klik menu New, kemudian buat sebuah database baru dengan nama laundry\_apps. Database ini akan digunakan sebagai tempat penyimpanan semua data yang terkait dengan aplikasi.
- ✚ Setelah database berhasil dibuat, klik database laundry\_apps tersebut. Selanjutnya, buat sebuah tabel baru dengan nama user. Tabel inilah yang akan digunakan untuk menyimpan data pengguna aplikasi (misalnya admin atau user).
- ✚ Ketika membuat tabel user, tentukan jumlah kolom sesuai kebutuhan, misalnya 4 kolom (id, name, username, password). Klik tombol Create.
- ✚ Setelah itu, akan muncul tampilan pengaturan struktur tabel seperti gambar di atas. Setiap kolom diisi dengan tipe data dan atribut yang sesuai:
  - id → bertipe INT, dijadikan PRIMARY KEY, serta diatur sebagai AUTO\_INCREMENT agar nilainya bertambah otomatis setiap ada data baru.
  - name → bertipe VARCHAR(128), digunakan untuk menyimpan nama lengkap user.
  - username → bertipe VARCHAR(128), digunakan untuk menyimpan username login.
  - password → bertipe TEXT, digunakan untuk menyimpan kata sandi user.

Dengan langkah ini, database laundry\_apps sudah memiliki tabel user yang siap digunakan sebagai penyimpanan data utama untuk fitur CRUD (Create, Read, Update, Delete) pada aplikasi Java yang sedang dibuat.

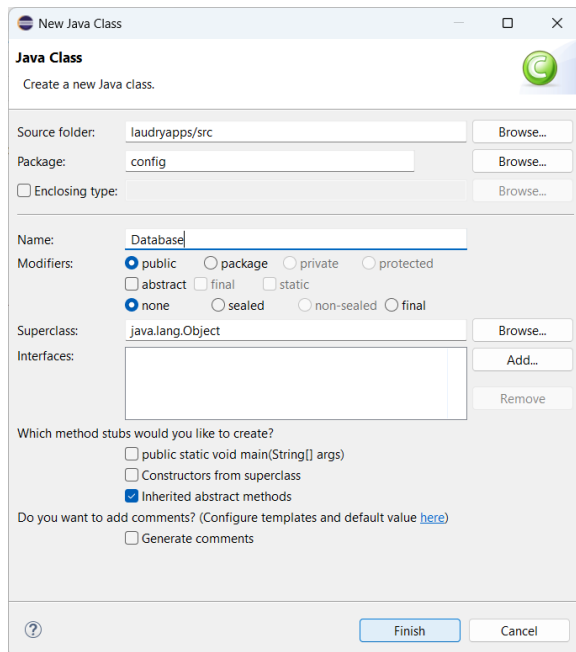
#### 4. Membuat Koneksi ke Database MySQL



Setelah berhasil menambahkan MySQL Connector pada project, tahap berikutnya adalah membuat koneksi ke database MySQL. Koneksi ini diperlukan agar aplikasi dapat melakukan proses pengolahan data, seperti menyimpan, menampilkan, memperbarui, maupun menghapus data dari basis data. Langkah pertama yang dilakukan adalah membuat sebuah package baru dengan nama config. Package ini berfungsi sebagai wadah untuk menyimpan konfigurasi aplikasi, terutama konfigurasi yang berkaitan dengan koneksi ke database. Selanjutnya, di dalam package tersebut dibuat sebuah class bernama Database. Class ini berisi kode program yang mengatur

koneksi ke MySQL dengan mendefinisikan **URL database, username, dan password**, serta menyediakan method untuk menginisialisasi koneksi.

5. Setelah membuat database laundryapps di localhost/phpMyAdmin, kita akan membuat koneksi ke database. Buat package config, lalu baut class baru dengan nama Database, lalu tulis code seperti berikut.



```
package config;

import java.sql.*;

public class Database {
    Connection conn;

    public static Connection koneksi() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost/laundry_apps", "root", ""
            );
            return conn;
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
            return null;
        }
    }
}
```

### Import Library

```
import java.sql.*;

import javax.swing.JOptionPane;
```

Baris ini berfungsi untuk mengimpor library yang dibutuhkan. Package java.sql.\* digunakan agar program dapat mengakses class dan interface JDBC, seperti Connection, DriverManager, dan lain-lain. Sedangkan javax.swing.JOptionPane



berfungsi untuk menampilkan pesan dialog jika terjadi kesalahan saat program dijalankan.

#### **Deklarasi Class dan Variabel**

```
public class Database {  
    Connection conn;  
}
```

Pada bagian ini, didefinisikan class Database dengan sebuah variabel conn bertipe Connection. Variabel ini digunakan untuk menyimpan objek koneksi ke database.

#### **Method koneksi()**

```
public static Connection koneksi() {  
    try {  
        Class.forName("com.mysql.cj.jdbc.Driver");  
        Connection conn = DriverManager.getConnection(  
            "jdbc:mysql://localhost/laundry_apps", "root", ""  
        );  
        return conn;  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, e);  
        return null;  
    }  
}
```

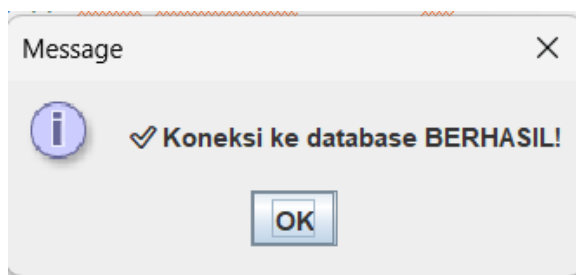
Method koneksi() bersifat static sehingga dapat dipanggil langsung tanpa harus membuat objek dari class Database. Pada bagian Class.forName("com.mysql.cj.jdbc.Driver"), driver MySQL dipanggil agar dapat digunakan oleh program. Kemudian, DriverManager.getConnection(...) digunakan untuk membuat koneksi ke database dengan URL jdbc:mysql://localhost/laundry\_apps, username root, dan password kosong. Apabila koneksi berhasil, method ini akan mengembalikan objek Connection. Namun, jika terjadi error, misalnya database tidak ditemukan atau driver tidak

tersedia, maka program akan menampilkan pesan kesalahan menggunakan JOptionPane dan mengembalikan nilai null.

## 6. Membuat kelas untuk mengecek koneksi

```
package config;
import java.sql.Connection;

public class TestKoneksi {
    public static void main(String[] args) {
        try {
            Connection c = Database.koneksi();
            if (c != null) {
                JOptionPane.showMessageDialog(null, "Koneksi ke database BERHASIL!");
            } else {
                JOptionPane.showMessageDialog(null, "Koneksi ke database GAGAL (Connection null).");
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "ERROR: " + e.getMessage());
            e.printStackTrace(); // biar kelihatan detail di Console
        }
    }
}
```



### Import Library

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

Library `java.sql.Connection` digunakan untuk membuat objek koneksi ke database, sedangkan `javax.swing.JOptionPane` berfungsi untuk menampilkan notifikasi dalam bentuk dialog pesan kepada pengguna.

### Class TestKoneksi dan Method main()

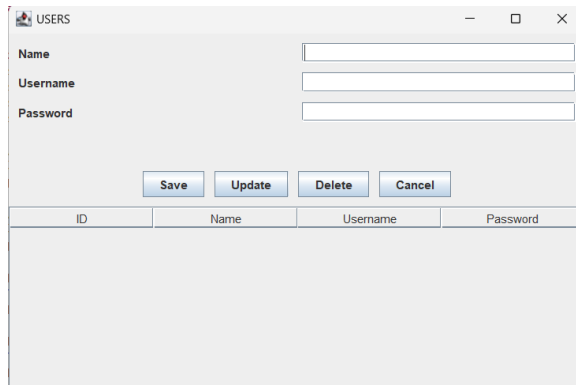
Pada bagian ini, program memanggil method `Database.koneksi()` untuk melakukan percobaan koneksi ke database. Jika objek `Connection` berhasil dibuat (tidak bernilai null), maka akan ditampilkan pesan "**Koneksi ke database BERHASIL**". Sebaliknya, jika objek `Connection` bernilai null, maka ditampilkan pesan "**Koneksi ke database GAGAL**".

### Penanganan Error

Apabila terjadi kesalahan, misalnya driver JDBC tidak ditemukan atau database tidak tersedia, program akan menampilkan pesan error menggunakan `JOptionPane`.

Selain itu, detail kesalahan juga ditampilkan di console melalui `e.printStackTrace()` untuk memudahkan proses debugging.

## 7. Membuat tampilan CRUD untuk User



Bagian ini merupakan langkah untuk membuat tampilan CRUD (Create, Read, Update, Delete) User menggunakan `JFrame` pada package `ui` dengan nama class `UserFrame`. Tampilan ini adalah antarmuka (GUI) yang akan digunakan untuk mengelola data user di dalam aplikasi.

### 🎨 Form Input Data

- **`JTextField txtName`** → digunakan untuk menginput *Name* dari user.
- **`JTextField txtUsername`** → digunakan untuk menginput *Username*.
- **`JTextField txtPassword`** → digunakan untuk menginput *Password*.

Ketiga text field ini menjadi komponen utama untuk memasukkan data user baru maupun mengedit data user yang sudah ada.

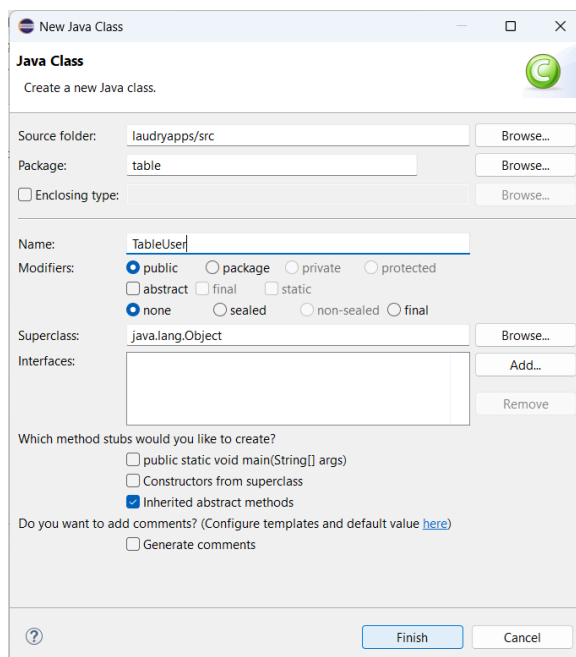
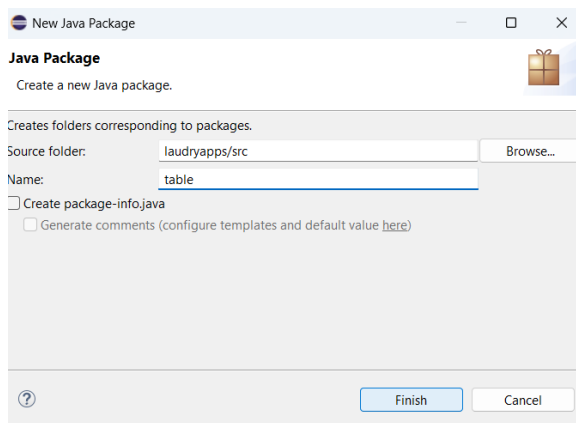
### 🎨 Tombol Aksi (CRUD Buttons)

- **`JButton btnSave`** → berfungsi untuk menyimpan data user baru ke dalam database.
- **`JButton btnUpdate`** → digunakan untuk memperbarui data user yang sudah ada.
- **`JButton btnDelete`** → digunakan untuk menghapus data user tertentu.
- **`JButton btnCancel`** → berfungsi untuk membatalkan input atau mengosongkan form.

Keempat tombol ini adalah implementasi dari fungsi CRUD yang nantinya akan dihubungkan ke logika database.

## Tabel Data

- **JTable tableUsers** → berfungsi untuk menampilkan seluruh data user yang sudah tersimpan di database. Data yang ditampilkan pada tabel dapat dipilih untuk kemudian diedit atau dihapus.
8. Selanjutnya kita membuat table model yang berfungsi untuk data dari database dan ditampilkan kedalam table. Dengan cara membaut package baru dengan nama table dan membuat class baru dengan nama tableuser. Lalu tuliskan code berikut.



```

package table;

import javax.swing.table.AbstractTableModel;

public class TableUser extends AbstractTableModel {
    private List<User> ls;
    private final String[] columnNames = {"ID", "Name", "Username", "Password"};

    public TableUser(List<User> ls) {
        this.ls = ls;
    }

    @Override
    public int getRowCount() {
        return ls.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public String getColumnName(int column) {
        return columnNames[column];
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        User user = ls.get(rowIndex);
        switch (columnIndex) {
            case 0: return user.getId();
            case 1: return user.getName();
            case 2: return user.getUsername();
            case 3: return user.getPassword();
            default: return null;
        }
    }
}

```

Kode di atas merupakan implementasi **class TableUser** yang dibuat untuk menampilkan data user ke dalam komponen **JTable** pada aplikasi Java Swing. Class ini ditempatkan dalam package **table** dan **meng-extend AbstractTableModel**, sehingga dapat digunakan sebagai model data untuk JTable.

#### Deklarasi Class dan Atribut

```

public class TableUser extends AbstractTableModel {
    private List<User> ls;
    private final String[] columnNames = {"ID", "Name", "Username",
    "Password"};

```

- Class TableUser mewarisi AbstractTableModel, yang mengharuskan untuk mengimplementasikan method-method tertentu seperti getRowCount(), getColumnCount(), dan getValueAt().
- Variabel ls bertipe List<User> digunakan untuk menyimpan kumpulan objek User yang akan ditampilkan pada tabel.
- Array columnNames menyimpan judul kolom tabel, yaitu *ID*, *Name*, *Username*, dan *Password*.

#### Konstruktor

```

public TableUser(List<User> ls) {
    this.ls = ls;
}

```

Konstruktor ini menerima parameter berupa `List<User>` dan menyimpannya ke dalam variabel `ls`. Data inilah yang nantinya akan ditampilkan di tabel.

#### **Method `getRowCount()`**

```
@Override  
public int getRowCount() {  
    return ls.size();  
}
```

Mengembalikan jumlah baris pada tabel sesuai dengan banyaknya data User di dalam list `ls`.

#### **Method `getColumnCount()`**

```
@Override  
public int getColumnCount() {  
    return columnNames.length;  
}
```

Mengembalikan jumlah kolom tabel, yang diambil dari panjang array `columnNames`. Pada program ini terdapat 4 kolom.

#### **Method `getColumnName()`**

```
@Override  
public String getColumnName(int column) {  
    return columnNames[column];  
}
```

Digunakan untuk memberikan judul pada kolom tabel berdasarkan index kolom.

#### **Method `getValueAt()`**

```
@Override  
public Object getValueAt(int rowIndex, int columnIndex) {  
    User user = ls.get(rowIndex);  
    switch (columnIndex) {  
        case 0: return user.getId();  
        case 1: return user.getNama();  
        case 2: return user.getUsername();  
        case 3: return user.getPassword();  
    }
```

```

        default: return null;
    }
}

```

Method ini mengatur data apa yang akan ditampilkan pada setiap sel tabel berdasarkan baris (rowIndex) dan kolom (columnIndex).

- Kolom 0 → ID user
- Kolom 1 → Name user
- Kolom 2 → Username
- Kolom 3 → Password

Setiap baris merepresentasikan satu objek User dari list.

## 9. Membuat fungsi DAO

The image shows two screenshots of IDE dialog boxes. The top dialog is 'New Java Package' with the following fields: Source folder: laudryapps/src, Name: DAO, and checkboxes for 'Create package-info.java' and 'Generate comments'. The bottom dialog is 'New Java Class' with the following fields: Source folder: laudryapps/src, Package: DAO, Name: UserDao, Modifiers: public (selected), Superclass: java.lang.Object, and checkboxes for 'Inherited abstract methods' (selected) and 'Generate comments'.

```
package DAO;

import java.util.List;

public interface UserDao {
    void save(User user);
    List<User> show();
    void delete(String id);
    void update(User user);
}
```

Kode di atas merupakan sebuah **interface** bernama UserDao yang ditempatkan dalam package **DAO (Data Access Object)**. Interface ini mendefinisikan **kontrak** atau blueprint operasi yang harus diimplementasikan untuk melakukan pengolahan data pada entitas User. Dengan kata lain, UserDao berisi deklarasi method untuk kebutuhan **CRUD (Create, Read, Update, Delete)** data User di database.

#### Import Library dan Model

```
import java.util.List;
import model.User;
```

- java.util.List digunakan untuk menampung kumpulan objek User dalam bentuk list.
- model.User menunjukkan bahwa interface ini bekerja dengan objek User dari package model.

#### Deklarasi Interface

```
public interface UserDao {
    void save(User user);
    List<User> show();
    void delete(String id);
    void update(User user);
}
```

Interface UserDao mendefinisikan empat operasi utama:

- **void save(User user)** → Method untuk menyimpan data user baru ke database.



- **List<User> show()** → Method untuk menampilkan semua data user yang tersimpan, dikembalikan dalam bentuk list.
- **void delete(String id)** → Method untuk menghapus data user berdasarkan id.
- **void update(User user)** → Method untuk memperbarui data user yang sudah ada.

## 10. Mendapatkan fungsi DAO

```
package DAO;

import java.sql.Connection;

public class UserRepo implements UserDao {
    private Connection connection;
    final String insert = "INSERT INTO user (name, username, password) VALUES (?, ?, ?)";
    final String select = "SELECT * FROM user";
    final String delete = "DELETE FROM user WHERE id=?";
    final String update = "UPDATE user SET name=?, username=?, password=? WHERE id=?";

    public UserRepo() {
        connection = Database.koneksi();
    }

    @Override
    public void save(User user) {
        // TODO Auto-generated method stub
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(insert);
            st.setString(1, user.getName());
            st.setString(2, user.getUsername());
            st.setString(3, user.getPassword());
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public List<User> show() {
        // TODO Auto-generated method stub
        List<User> ls = null;
        try {
            ls = new ArrayList<User>();
            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery(select);
            while (rs.next()) {
                User user = new User();
                user.setId(rs.getString("id"));
                user.setName(rs.getString("name"));
                user.setUsername(rs.getString("username"));
                user.setPassword(rs.getString("password"));
                ls.add(user);
            }
        } catch (SQLException e) {
            Logger.getLogger(UserDao.class.getName()).log(Level.SEVERE, null, e);
        }
        return ls;
    }

    @Override
    public void delete(String id) {
        // TODO Auto-generated method stub
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(delete);
            st.setString(1, id);
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```

@Override
public void update(User user) {
    // TODO Auto-generated method stub
    PreparedStatement st = null;
    try {
        st = connection.prepareStatement(update);
        st.setString(1, user.getNama());
        st.setString(2, user.getUsername());
        st.setString(3, user.getPassword());
        st.setString(4, user.getId());
        st.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            st.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Kelas **UserRepo** merupakan implementasi dari interface **UserDao** yang berfungsi sebagai penghubung antara aplikasi dengan database. Pada kelas ini digunakan **JDBC (Java Database Connectivity)** untuk melakukan operasi **CRUD** (Create, Read, Update, Delete) terhadap tabel user. Setiap method di dalamnya merepresentasikan aksi terhadap database, mulai dari menyimpan data baru, menampilkan seluruh data, memperbarui data yang sudah ada, hingga menghapus data tertentu.

#### **Method save(User user)**

Method ini digunakan untuk **menyimpan data user baru** ke dalam tabel user. Prosesnya dimulai dengan menyiapkan pernyataan SQL INSERT, kemudian parameter diisi dengan data dari objek User seperti name, username, dan password. Setelah itu, perintah dijalankan menggunakan `executeUpdate()`. Method ini akan menambahkan baris baru di database sesuai input dari user.

#### **Method show()**

Method ini digunakan untuk **menampilkan semua data user** yang ada di dalam tabel. Prosesnya dilakukan dengan mengeksekusi query `SELECT * FROM user`. Hasil query kemudian disimpan dalam objek `ResultSet`, lalu setiap baris data dikonversi menjadi objek `User` dan dimasukkan ke dalam list. Hasil akhirnya berupa `List<User>` yang berisi kumpulan data user dari database.

#### **Method delete(String id)**

Method ini berfungsi untuk **menghapus data user berdasarkan ID**. Query yang digunakan adalah `DELETE FROM user WHERE id=?`. Parameter ID diisi sesuai data yang dipilih, lalu perintah `executeUpdate()` dijalankan untuk menghapus baris

data dari tabel user. Dengan demikian, hanya data dengan ID tertentu yang akan dihapus dari database.

#### **Method update(User user)**

Method ini digunakan untuk **memperbarui data user yang sudah ada** di database. Query yang digunakan adalah UPDATE user SET name=?, username=?, password=? WHERE id=?. Data yang baru akan menggantikan data lama berdasarkan ID. Parameter diisi dari atribut pada objek User, kemudian dijalankan dengan executeUpdate(). Hasilnya, data user pada tabel akan berubah sesuai input terbaru.

## **D. KESIMPULAN**

Berdasarkan hasil praktikum, dapat disimpulkan bahwa proses menghubungkan aplikasi Java dengan database MySQL membutuhkan konfigurasi yang tepat, mulai dari menambahkan MySQL Connector hingga membuat database dan tabel melalui phpMyAdmin. Dengan adanya koneksi tersebut, aplikasi dapat melakukan operasi dasar pengolahan data (CRUD) seperti menambahkan, menampilkan, memperbarui, dan menghapus data. Hal ini menunjukkan pentingnya pemahaman konsep JDBC sebagai jembatan komunikasi antara aplikasi Java dan database.

Selain itu, penerapan JTable dengan AbstractTableModel memudahkan penyajian data secara terstruktur dalam aplikasi berbasis GUI. Mahasiswa juga dapat memahami bagaimana alur kerja program dari mulai input data, penyimpanan ke database, hingga menampilkannya kembali ke layar aplikasi. Praktikum ini menjadi dasar penting dalam pembuatan aplikasi yang lebih kompleks, karena mampu melatih pemahaman mahasiswa tentang integrasi Java dengan database serta penerapan prinsip pemrograman berorientasi objek dalam membangun sistem informasi sederhana.