

Modul 14

Aplikasi yang dibuat merupakan app berbasis desktop dengan menggunakan bahasa Java. Beberapa hal yang perlu diperhatikan:

- Menggunakan swing untuk komponen GUI
- Menggunakan MariaDB dari LAMPP untuk penyimpanan data
- Menggunakan Netbeans 12

GUI yang ditampilkan adalah sebagai berikut:

The screenshot shows a Java Swing window titled "Formulir" with standard window controls. Inside, the title "Formulir Data Pelanggan" is centered. Below it, a section titled "Data Pelanggan" contains a form with the following fields:

- Id Member:** A text input field.
- Nama:** A text input field.
- Tahun Lahir:** A dropdown menu with "1990" selected.
- Jenis Member:** A dropdown menu with "silver" selected.

Below the form are two buttons: "Simpan" and "Reset". At the bottom of the window is a "Hapus" button.

Below the form, there is a table displaying customer data:

Id Member	Nama	Tahun Lahir	Jenis
MEM-001	Melody Nurra...	1992	gold
MEM-002	Sonya Panda...	1996	gold
MEM-003	Nabilah Ratna...	1999	platinum
MEM-005	Naomi	1993	silver

Keterangan tambahan:

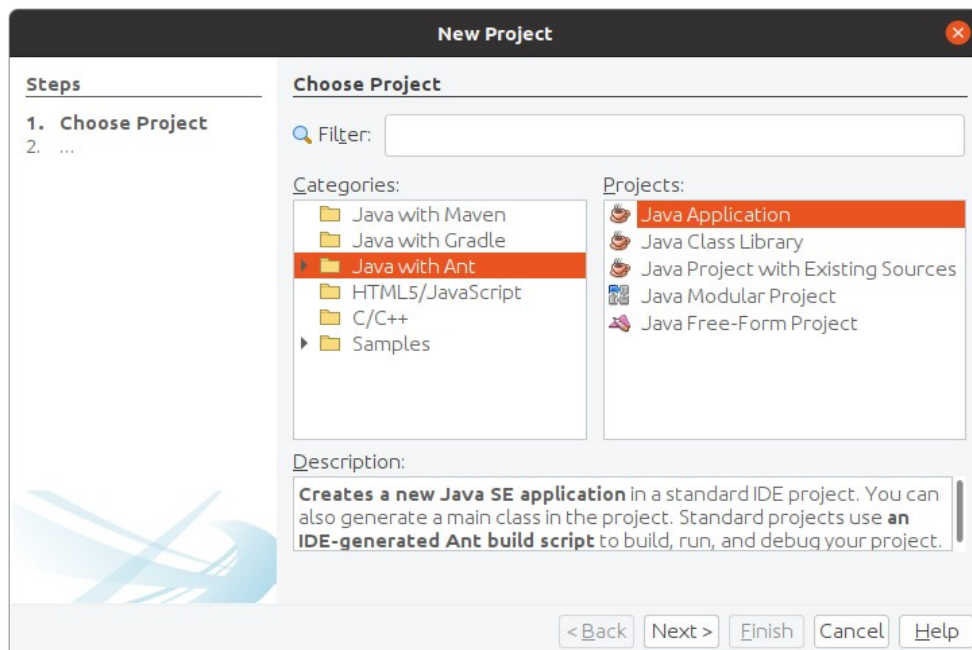
- Tahun lahir ditentukan 1990-2000
- Jenis member: silver, gold, platinum
- Operasi: insert, delete, select
- Id member diberikan manual (tidak auto increment)
- Terdapat pop-up message ketika berhasil/gagal insert/delete
- Terdapat pop-up confirmation message ketika akan menghapus data

Contoh pengerjaan dari pembuatan app merupakan bentuk sederhana dari pengaksesan data MySQL pada localhost. Cara yang digunakan tidak memakai DAO atau sejenisnya.

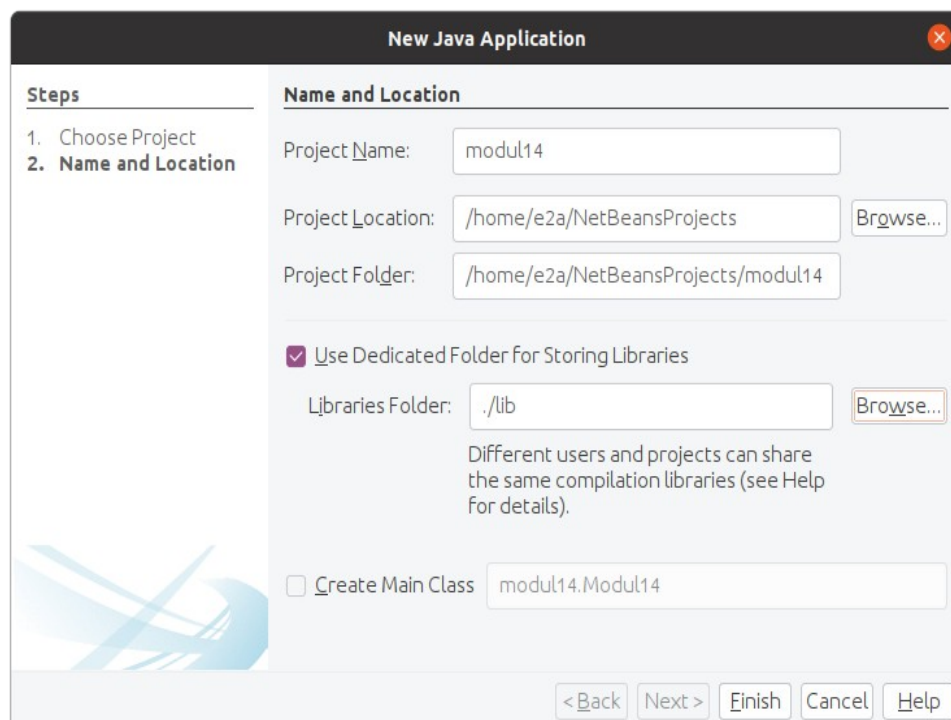
Contoh project dapat dilihat di <https://github.com/golchafun/swing-mysql-contoh>

Langkah 1: Buat project netbeans

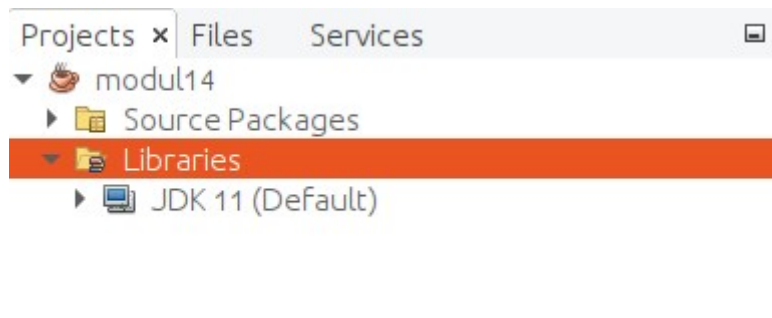
Buat project dengan categories “Java with Ant”.



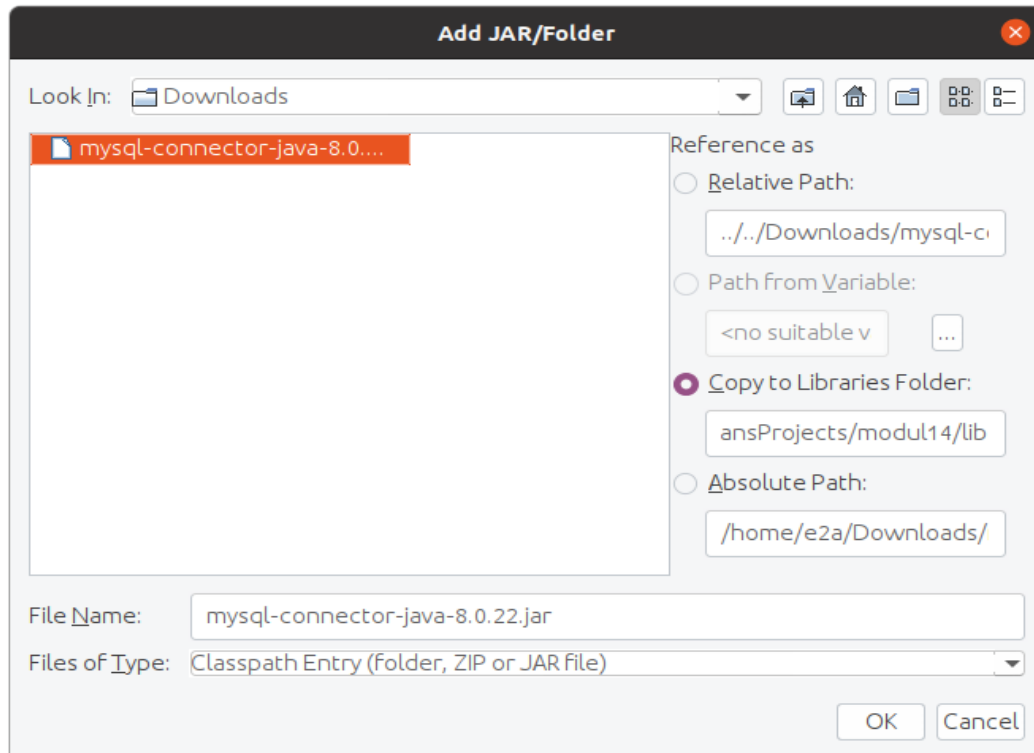
Berikan nama project dan lokasi penyimpanan sesuai keinginan. Pastikan mencentang pilihan “Use Dedicated Folder for Storing Libraries”



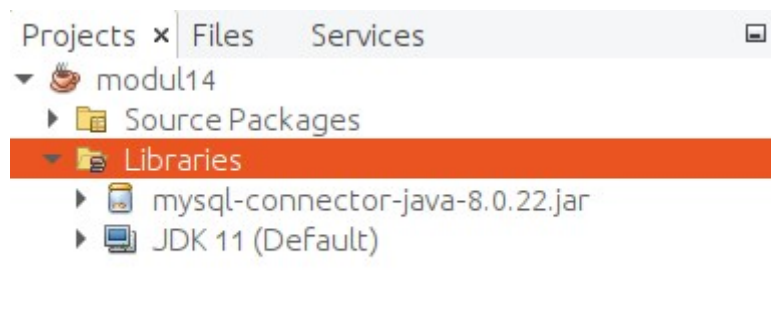
Tambahkan library jdbc dengan klik kanan pada folder “Libraries” → “Add JAR/folder”



Tambahkan driver jdbc, pastikan memilih pilihan “Copy to Libraries Folder”.

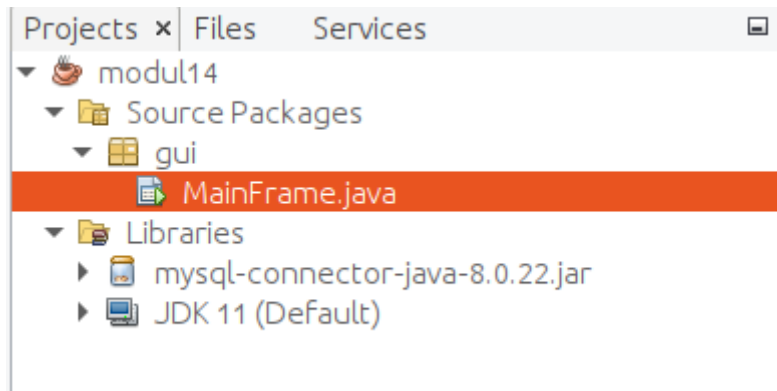


Pastikan telah ada driver di bagian libraries.

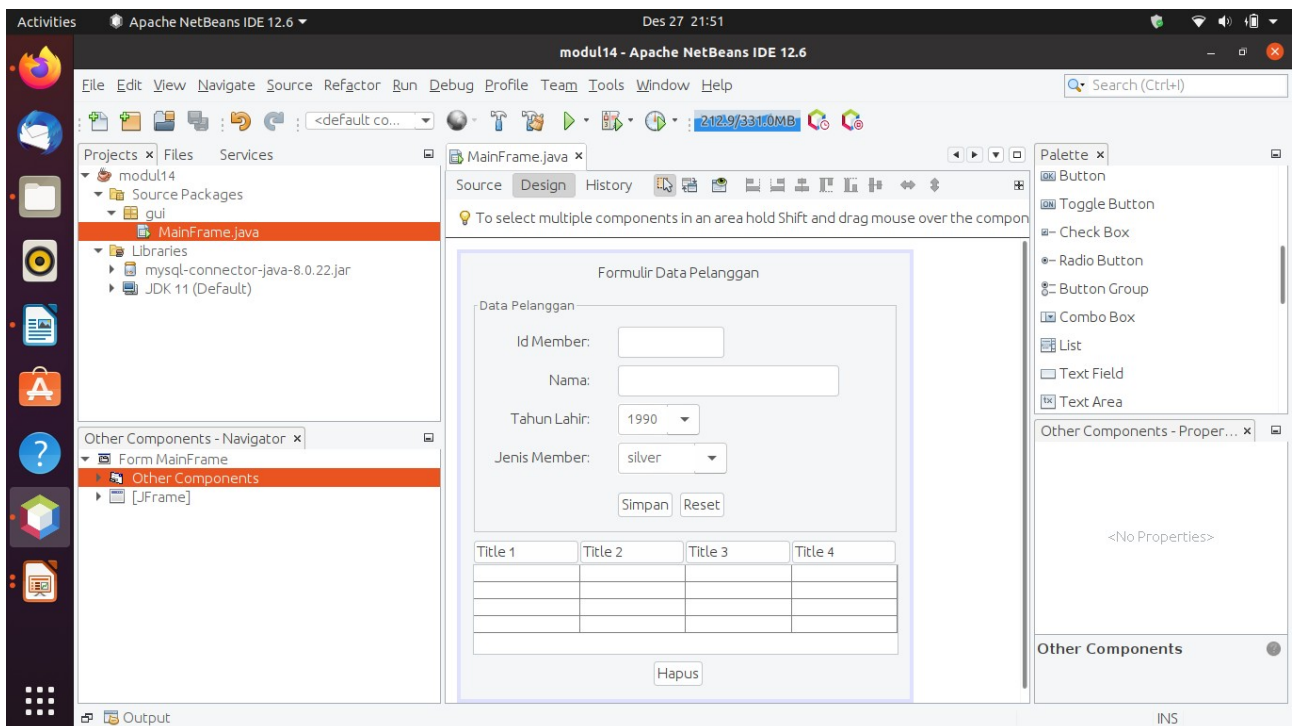


Langkah 2: Buat & Atur Komponen GUI

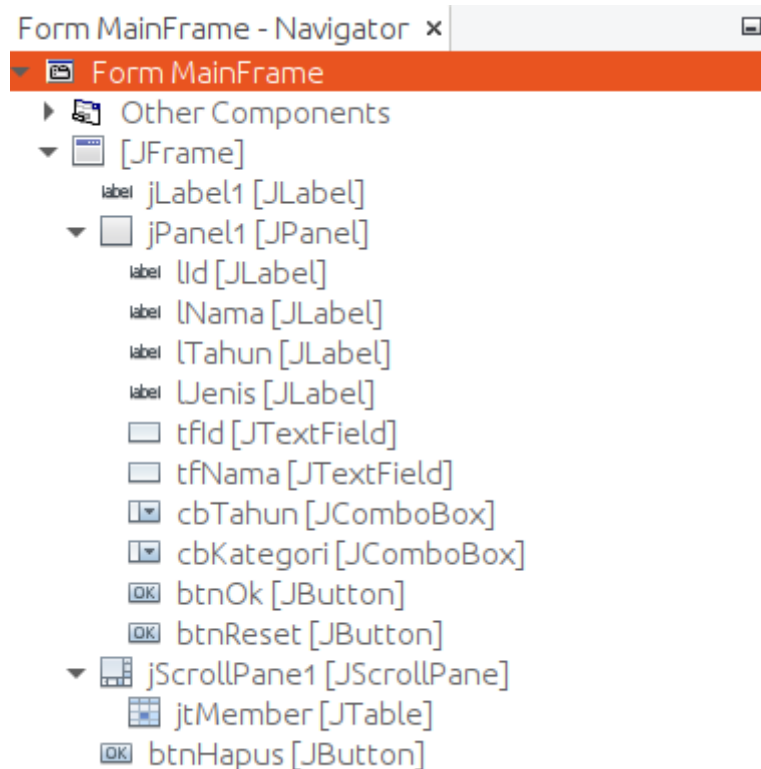
Buat package baru bernama “gui”, lalu pada package tersebut tambahkan sebuah JFrame (nama bebas, tapi di sini dituliskan MainForm).).



Desain frame sesuai keinginan/keperluan tugas.



Atur nama component sebagai berikut (hal ini berpengaruh pada kode yang ditulis nantinya).



Jalankan app, pastikan tampil sesuai keinginan

Formulir

Formulir Data Pelanggan

Data Pelanggan

Id Member:

Nama:

Tahun Lahir:

Jenis Member:

Title 1	Title 2	Title 3	Title 4

Tampilkan GUI di tengah layar dengan kode “setLocationRelativeTo(null);” dituliskan pada konstruktor.

```
public MainFrame() {  
    initComponents();  
    setLocationRelativeTo(null);  
}
```

Berikan aksi pada tombol “Reset” dengan membuat method reset(), dan panggil method pada aksi button.

```
private void btnResetActionPerformed(java.awt.event.ActionEvent evt) {  
    reset();  
}  
  
private void reset(){  
    tfId.setText("");  
    tfNama.setText("");  
    cbKategori.setSelectedIndex(0);  
    cbTahun.setSelectedIndex(0);  
}
```










Tes app, pastikan fungsi tombol reset berjalan dengan baik.

Langkah 3: Buat Database & Isinya

Buat sebuah database bernama **pbo_modul14** dan sebuah tabel bernama **pelanggan**.

#	Name	Type
1	id_member 	varchar(10)
2	nama	varchar(50)
3	tahun_lahir	int(11)
4	kategori	varchar(10)

Isi tabel tersebut sesuai keinginan atau keperluan tugas.

		id_member	nama	tahun_lahir	kategori
<input type="checkbox"/>	 Edit	 Copy	 Delete	MEM-001	Melody Nurramdhani 1992 gold
<input type="checkbox"/>	 Edit	 Copy	 Delete	MEM-002	Sonya Pandawarman 1996 gold
<input type="checkbox"/>	 Edit	 Copy	 Delete	MEM-003	Nabilah Ratna Ayu 1999 platinum

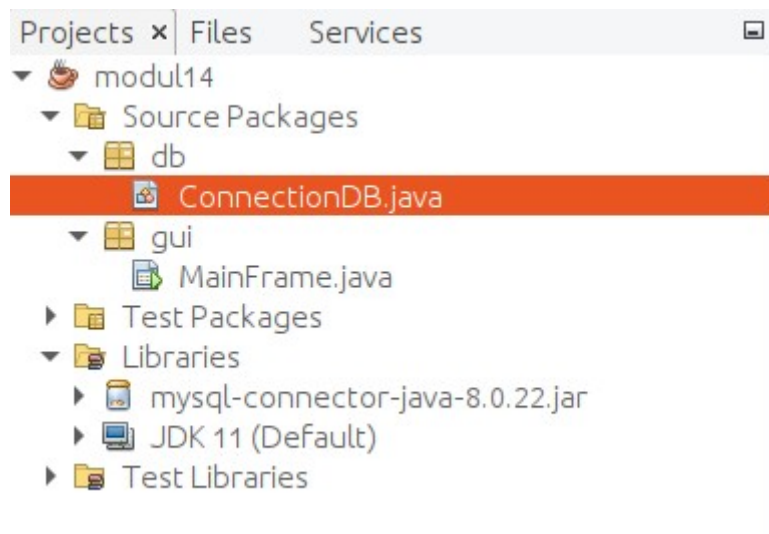
```
INSERT INTO `pelanggan` (`id_member`, `nama`, `tahun_lahir`, `kategori`) VALUES ('MEM-001', 'Melody Nurramdhani', '1992', 'gold');
```

```
INSERT INTO `pelanggan` (`id_member`, `nama`, `tahun_lahir`, `kategori`) VALUES ('MEM-002', 'Sonya Pandawarman', '1996', 'gold');
```

```
INSERT INTO `pelanggan` (`id_member`, `nama`, `tahun_lahir`, `kategori`) VALUES ('MEM-003', 'Nabilah Ratna Ayu', '1999', 'platinum');
```

Langkah 4: Akses Database

Pada project Netbeans, buat sebuah package bernama “db”, dan bentuk sebuah class bernama bebas (contoh pada tulisan ini bernama ConnectionDB).



Pada method di class ini, lakukan pembuatan objek Connection, dan kembalikan objek tersebut.

```
package db;
```

```
import com.mysql.cj.jdbc.MySQLDataSource;  
import java.sql.Connection;  
import java.sql.SQLException;
```

```
public class ConnectionDB {  
    public static Connection createConnection() throws SQLException {  
        MySQLDataSource dataSource = new MySQLDataSource();  
        String DB_URL="jdbc:mysql://localhost:3306/pbo_modul14?serverTimezone=Asia/Jakarta" ;  
        String DB_USERNAME="root";  
        String DB_PASSWORD= "";  
  
        dataSource.setUrl(DB_URL);  
        dataSource.setUser(DB_USERNAME);  
        dataSource.setPassword(DB_PASSWORD);  
  
        Connection conn = dataSource.getConnection();  
        return conn;  
    }  
}
```

Optional: method di atas dapat dites terlebih dahulu menggunakan main method sebagai berikut.

```
public static void main(String[] args) {  
    try {
```



```

        createConnection();
        System.out.println("Berhasil");
    } catch (SQLException ex) {
        System.out.println("error: "+ex.getMessage());
        System.out.println("Gagal");
    }
}

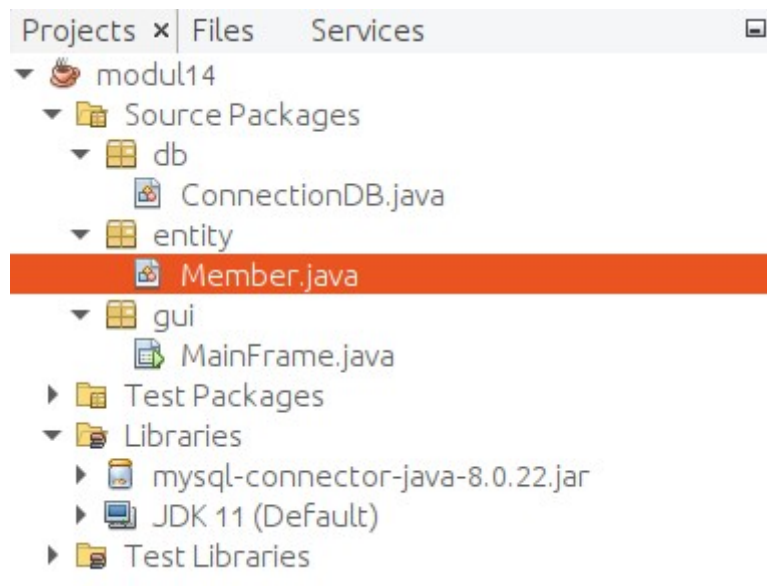
```

Jika keluar tulisan “Berhasil”, berarti OK. Jika tidak, lakukan penyesuaian/perbaikan terhadap error yang ditampilkan.

Main method ini dapat dihapus jika sudah mengeluarkan tulisan “Berhasil”.

Langkah 5: Buat Entitas Class

Setelah membuat class yang mengakses database, buat package bernama entity dan class bernama “Member” untuk menampung data member dari database.



Tuliskan kode class member mewakili data dari database.

```
package entity;
```

```
public class Member {
```

```
    private String id, nama, kategori;
    private int tahunLahir;
```

```
    public Member(String id, String nama, int tahunLahir, String kategori) {
        this.id = id;
        this.nama = nama;
        this.kategori = kategori;
        this.tahunLahir = tahunLahir;
    }
}

```

```
public String getIdMember() {  
    return id;  
}  
  
public String getNama() {  
    return nama;  
}  
  
public String getKategori() {  
    return kategori;  
}  
  
public int getTahunLahir() {  
    return tahunLahir;  
}  
  
public Object[] toObjects() {  
    Object[] o = {id, nama, tahunLahir, kategori};  
    return o;  
}  
}
```

Langkah 6: Akses Data & Tampilkan

Berikutnya, beralih ke class MainFrame. Pada class MainFrame, akses method `createConnection` dari class `ConnectionDB`. Method ini akan digunakan untuk mengakses isi dari database dan menampilkannya ke `JTable` "jtMember".

Buat sebuah method untuk mengambil data pelanggan dari database untuk dimasukkan ke objek class `Member` yang disimpan dalam array list of `Member` (bernama `members`).

```
private List<Member> fetchMembers() {
    List<Member> members = new ArrayList();
    try {
        Connection conn = ConnectionDB.createConnection();
        String kueri = "SELECT * FROM pelanggan";
        PreparedStatement ps = conn.prepareStatement(kueri);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            members.add(new Member(rs.getString("id_member"),
                                    rs.getString("nama"),
                                    rs.getInt("tahun_lahir"),
                                    rs.getString("kategori")));
        }
    } catch (SQLException ex) {
        System.out.println("Err: " + ex.getMessage());
    }
    return members;
}
```

Pastikan melakukan import dari class terkait, yaitu:

```
import db.ConnectionDB;
import entity.Member;
import java.util.List;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import javax.swing.table.DefaultTableModel;
```

Berikutnya, buat method untuk mem-populate table berdasarkan data yang di-return oleh method sebelumnya, `fetchMembers()`.

```
private void populateTable() {
    DefaultTableModel model = (DefaultTableModel) jtMember.getModel();
    model.setRowCount(0); model.setColumnCount(0);
    model.addColumn("Id Member");
    model.addColumn("Nama");
    model.addColumn("Tahun Lahir");
    model.addColumn("Jenis");
}
```

```

List<Member> members = fetchMembers();
for(Member m:members){
    model.addRow(m.toObjects());
}
}

```

Panggil method tersebut pada konstruktor class MainFrame.java

```

public MainFrame() {
    initComponents();
    setLocationRelativeTo(null);
    populateTable();
}

```

Jalankan app, dan pastikan app sudah menampilkan data dalam tabel.

Id Member	Nama	Tahun Lahir	Jenis
MEM-001	Melody Nurra...	1992	gold
MEM-002	Sonya Panda...	1996	gold
MEM-003	Nabilah Ratna...	1999	platinum

Hal yang perlu diperhatikan dari cara load data di atas adalah semua data diambil dari database untuk ditampilkan ke dalam jTable. Hal ini tidak bermasalah jika datanya kecil, akan tetapi menjadi permasalahan jika data yang ditampilkan sangat banyak.

Langkah 7: Simpan Data

Tambahkan aksi pada tombol simpan. Cek apakah data yang sudah diberikan sudah valid (teks field tidak kosong). Pastikan proses simpan ke database berlangsung hanya ketika data sudah valid.

Langkah awal, buat sebuah method cekData() pada MainFrame.java untuk mengecek validitas input pengguna.

```
private boolean cekData() {
    if (tfId.getText().isEmpty() || tfNama.getText().isEmpty()) {
        return false;
    }
    return true;
}
```

Berikutnya, buat sebuah method untuk melakukan penyimpanan.

```
private boolean simpan(Member m) {
    try {
        Connection conn = ConnectionDB.createConnection();
        String kueri = "INSERT INTO pelanggan(id_member, nama, tahun_lahir, kategori) "
            + "VALUES (?, ?, ?, ?)";
        PreparedStatement ps = conn.prepareStatement(kueri);
        ps.setString(1, m.getIdMember());
        ps.setString(2, m.getNama());
        ps.setInt(3, m.getTahunLahir());
        ps.setString(4, m.getKategori());
        int hasil = ps.executeUpdate();
        if (hasil > 0) {
            return true;
        }
    } catch (SQLException ex) {
        System.out.println("Err: " + ex.getMessage());
    }
    return false;
}
```

Panggil kedua method pada aksi button penyimpanan.

```
private void btnOkActionPerformed(java.awt.event.ActionEvent evt) {
    if (!cekData()) {
        JOptionPane.showMessageDialog(null,
            "Input Kosong",
            "Kesalahan",
            JOptionPane.ERROR_MESSAGE);
    } else {
```

```

String id = tfId.getText();
String nama = tfNama.getText();
int tahun = Integer.parseInt(cbTahun.getSelectedItem().toString());
String jenis = cbKategori.getSelectedItem().toString();
Member m = new Member(id, nama, tahun, jenis);
if (simpan(m)) {
    JOptionPane.showMessageDialog(null, "Berhasil Insert");
    populateTable();
} else {
    JOptionPane.showMessageDialog(null, "Gagal Insert");
}
reset();
}
}

```

Hal yang perlu diperhatikan, penyimpanan di atas akan mem-populate data ke dalam table dengan menghapus data sebelumnya, dan melakukan pengambilan data kembali sebanyak data yang tersimpan. Cara ini tidak terlalu efisien walaupun dapat digunakan untuk data yang sedikit (sekitar $n < 50$). Akan lebih baik untuk mendeteksi hanya perubahan data saja dan menampilkannya.

Pastikan tidak ada error code, jalankan app, dan pastikan fungsi simpan berjalan sesuai yang seharusnya.

Langkah 8: Hapus Data

Pada hapus data, hal yang dilakukan pertama adalah membuat method untuk melakukan hapus data berdasarkan id yang diterima.

```

private boolean hapus(String id) {
    try {
        Connection conn = ConnectionDB.createConnection();
        String kueri = "DELETE FROM pelanggan WHERE id_member = ?";
        PreparedStatement ps = conn.prepareStatement(kueri);
        ps.setString(1, id);
        int hasil = ps.executeUpdate();
        if (hasil > 0) {
            return true;
        }
    } catch (SQLException ex) {
        System.out.println("Err: " + ex.getMessage());
    }
    return false;
}

```

Berikutnya, pada aksi button hapus, ambil id dengan mengecek row terpilih dari jTable. Lalu, dapatkan value id dari posisi (row,0) yang menyatakan posisi baris terpilih & kolom pertama dari model tabel. Nilai id akan dilewatkan pada pemanggilan method sebelumnya untuk melakukan penghapusan data.

Dikarenakan pengguna dapat menekan tombol hapus tanpa perlu memilih data dari tabel yang bukan merupakan skenario hapus, kode hapus dapat dimasukkan dalam blok try-catch. Sehingga memunculkan pesan kesalahan bahwa pengguna harus memilih data pada tabel sebelum melakukan hapus data.

```
private void btnHapusActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        //dapatkan id data yang akan dihapus  
        int selectedRow = jtMember.getSelectedRow();  
        DefaultTableModel model = (DefaultTableModel) jtMember.getModel();  
        String id = model.getValueAt(selectedRow, 0).toString();  
        //lakukan confirm  
        int pil = JOptionPane.showConfirmDialog(null, "Hapus data ID: " + id + "?");  
        if (pil == JOptionPane.YES_OPTION) {  
            if (hapus(id)) {  
                JOptionPane.showMessageDialog(null, "Hapus Berhasil");  
                populateTable();  
            } else {  
                JOptionPane.showMessageDialog(null, "Hapus Gagal");  
            }  
        }  
    } catch (ArrayIndexOutOfBoundsException ex) {  
        JOptionPane.showMessageDialog(null, "Pilih data pada tabel");  
    }  
}
```

Pastikan tidak ada error code, jalankan app, cek fungsi hapus dan pastikan berjalan sesuai ekspektasi.