GUI dan Database

1. Kompetensi

Setelah menempuh materi percobaan ini, mahasiswa mampu:

1. Menggunakan paradigma berorientasi objek untuk interaksi dengan database
2. Membuat Graphical User Interface (GUI)

2. Pendahuluan

Kali ini kita akan menggunakan paradigma berorientasi objek yang telah kita pelajari untuk membuat aplikasi yang melakukan manipulasi data di database melalui Graphical User Interface (GUI).

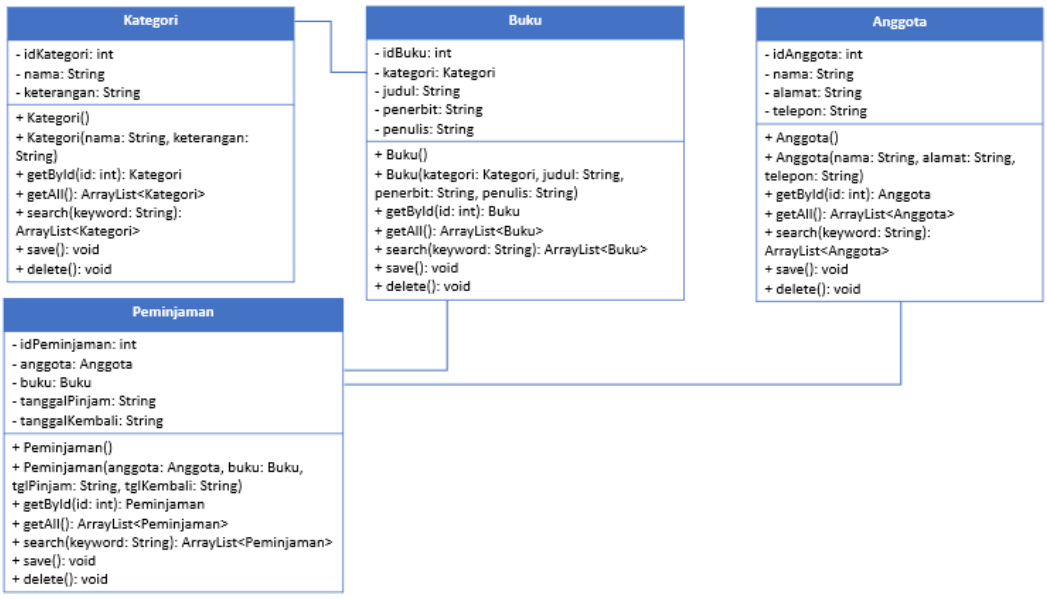
Secara umum, tahapan yang akan kita lakukan adalah sebagai berikut:

1. Membuat database yang berisi tabel-tabel yang diperlukan.
2. Membuat backend yang berisi class-class yang mewakili data yang ada pada database, dan class helper untuk melakukan eksekusi query database.
3. Membuat frontend sebagai antarmuka dengan pengguna. Frontend ini bisa berbasis teks (console), GUI, web, mobile, dan sebagainya.

Library yang digunakan untuk project ini antara lain:

1. JDBC (Java Data Base Connectivity), untuk melakukan interaksi ke database.
2. ArrayList, untuk menampung data hasil query ke database.
3. Swing, untuk membuat GUI.

Untuk percobaan, kita akan membuat sistem informasi Perpustakaan, yang memiliki data antara lain: Buku, Kategori, Anggota dan Peminjaman. Fitur yang ada pada aplikasi ini adalah anggota dapat melakukan peminjaman dan pengembalian buku. Berikut adalah class diagram untuk sistem informasi ini:

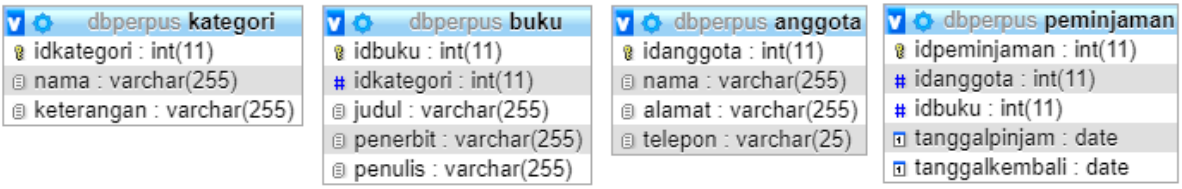


Dapat dilihat dari class diagram di atas, terdapat relasi antar class. Class Buku berelasi dengan Kategori dikarenakan terdapat atribut bertipe data Kategori di dalam class Buku. Demikian juga class Peminjaman yang berelasi dengan class Buku dan Anggota.

3. Percobaan

3.1 Percobaan 1 – Membuat Database

1. Langkah pertama untuk percobaan ini adalah membuat database. Install XAMPP, buka phpMyAdmin, buat database **dbperpus**, dan tabel-tabelnya:



2. Set semua primary key id pada tiap tabel (idanggota, idkategori, idpeminjaman, idbuku) dengan **Auto Increment**.
3. Buat foreign key antartable

3.2 Percobaan 2 – Mempersiapkan Project

1. Buat project baru, beri nama **Perpustakaan**.
2. Pada project explorer, klik kanan pada Libraries → Add Library, pilih MySQL JDBC Driver.
3. Jika tidak ada pilihan MySQL JDBC Driver:
    a. Buka link https://dev.mysql.com/downloads/connector/j/
    b. Pilih “Platform Independent”pada opsi Operating System
    c. Download ZIP Archive kemudian extract
    d. Pada project yang dibuat, klik kanan pada Libraries, kemudian add JAR/Folder… dan pilih file jar yang telah diextract sebelumnya
4. Buat package **frontend** dan **backend**. Cara membuat package adalah, pada project explorer, klik kanan pada Source Packages → New → Java Package, beri nama package nya (frontend, backend).

3.3 Percobaan 3

Membuat class helper untuk mengeksekusi query SQL.

1. Pada package **backend**, buat class **DBHelper**.
2. Import java.sql.*
3. Berikut adalah kode dari class DBHelper. Sesuaikan nilai url, user, password, dan nama database pada method **bukaKoneksi()** dengan setting database yang terinstall di sistem. Namun jika XAMPP diinstall dengan konfigurasi default, maka setting ini tidak perlu diubah.

    Beberapa istilah dalam JDBC:
    a. **DriverManager**: class yang mengelola driver
    b. **Driver**: interface yang menangani mekanisme pengaksesan database
    c. **Connection**: interface yang menangani koneksi (session) ke database tertentu
    d. **Statement**: interface yang menangani eksekusi SQL statement/query
    e. **ResultSet**: interface yang menampung hasil eksekusi query

Class DBHelper memiliki beberapa method, yaitu:

a. bukaKoneksi() untuk membuka koneksi ke database
b. insertQueryGetId() untuk mengeksekusi insert data ke database dan mengembalikan nilai id data yang baru dibuat
   ➔ method **executeUpdate(query, Statement.RETURN_GENERATED_KEYS)** digunakan untuk mengeksekusi query dan menginfomasikan driver untuk mengembalikan nilai key yang dihasilkan secara otomatis (melalui auto increment)
c. executeQuery() untuk mengeksekusi suatu SQL Query. Method ini mengembalikan nilai true jika query berhasil diekseksui dan false jika tidak berhasil
   ➔ method **executeUpdate()** digunakan untuk mengeksekusi query yang tidak mengembalikan ResultSet seperti INSERT, UPDATE, dan DELETE serta DDL (Data Definition Language) lainnya
d. selectQuery() di dalamnya memuat method **executeQuery()** yang mengeksekusi query dan mengembalikan ResultSet

```
package backend;

import java.sql.*;

public class DBHelper {
    private static Connection connection;
    private static ResultSet rs;
    private static Statement stmt;

    public static void openConnection() {
        if (connection == null) {
            try {
                String url = "jdbc:mysql://localhost:3306/perpustakaan";
                String user = "root";
                String password = "";
                Driver driver = new com.mysql.cj.jdbc.Driver();
                DriverManager.registerDriver(driver);
                connection = DriverManager.getConnection(url, user, password);
            } catch (SQLException t) {
                System.out.println("Error koneksi!");
            }
        }
    }

    public static ResultSet selectQuery(String query) {
        openConnection();

        try {
            stmt = connection.createStatement();
            rs = stmt.executeQuery(query);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return rs;
    }

    public static int insertQueryGetId(String sql) {
        openConnection();
        int result = -1;

        try {
            Statement stmt = connection.createStatement();
            stmt.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);
            ResultSet rs = stmt.getGeneratedKeys();

            if (rs.next()) {
```

```
                result = rs.getInt(1);
            }

            rs.close();
            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
            result = -1;
        }

        return result;
    }

    public static boolean executeQuery(String query) {
        openConnection();
        boolean result = false;

        try {
            Statement stmt = connection.createStatement();
            stmt.executeUpdate(query);
            result = true;

            stmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return result;
    }
}
```

3.4 Percobaan 4

Membuat class **Kategori** untuk menghandle CRUD pada tabel kategori.

1. Pada package **backend**, buat class baru yaitu **Kategori**.
2. Tambahkan import java.util.ArrayList dan java.sql.*

```
import java.util.ArrayList;
import java.sql.*;
```

3. Tambahkan atribut sesuai field pada tabel kategori.

```
    private int idkategori;
    private String nama;
    private String keterangan;
```

4. Tambahkan getter setter untuk setiap atribut. Anda bisa gunakan fasilitas **Insert Code** pada NetBeans. Caranya adalah, klik kanan sembarang tempat di editor, pilih Insert Code, pilih Setter and Getter, lalu pilih atributnya.

5. Tambahkan konstruktor default dan konstruktor custom yang digunakan untuk mengeset atribut nama dan keterangan. Atribut idkategori tidak boleh diset, karena id ini akan digenerate secara otomatis lewat fitur AutoIncrement pada MySQL.

```
public Kategori()
{

}

public Kategori(String nama, String keterangan)
{
    this.nama = nama;
    this.keterangan = keterangan;
}
```

Konstruktor default

Konstruktor custom

6. Method selectQuery() akan mengembalikan ResultSet dengan id yang sesuai. Selanjutnya dibuat objek Kategori baru berdasarkan data dari ResultSet tersebut.

```
public static Kategori getById(int id){
    Kategori kat = null;
    ResultSet rs = DBHelper.selectQuery("SELECT * FROM kategori "
                                        + "WHERE idkategori = " + id + "");

    try {
        while (rs.next()) {
            kat = new Kategori();
            kat.setIdkategori(rs.getInt("idkategori"));
            kat.setNama(rs.getString("nama"));
            kat.setKeterangan(rs.getString("keterangan"));
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return kat;
}
```

7. Tambahkan method **getAll()** untuk mendapatkan semua data Kategori yang ada di database. Method selectQuery() akan mengembalikan data dari table Kategori. Selanjutnya setiap data dari ResultSet akan dibuat objek kategori baru yang ditampung oleh listKategori bertipe ArrayList<Kategori>.

```
public static ArrayList<Kategori> getAll(){
    ArrayList<Kategori> listKategori = new ArrayList();
    ResultSet rs = DBHelper.selectQuery("SELECT * FROM kategori");

    try {
        while (rs.next()) {
            Kategori kat = new Kategori();
            kat.setIdkategori(rs.getInt("idkategori"));
            kat.setNama(rs.getString("nama"));
            kat.setKeterangan(rs.getString("keterangan"));

            listKategori.add(kat);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return listKategori;
}
```

8. Tambahkan method **search()** agar bisa melakukan pencarian data. Method ini mirip dengan method **getAll()** dengan tambahan filter berdasarkan keyword yang diinginkan.

```
public static ArrayList<Kategori> search(String keyword){
    ArrayList<Kategori> listKategori = new ArrayList();

    String query = "SELECT * FROM kategori"
                 + " WHERE nama LIKE '%" + keyword + "%'"
                 + " OR keterangan LIKE '%" + keyword + "%'";

    ResultSet rs = DBHelper.selectQuery(query);

    try {
        while (rs.next()) {
            Kategori kat = new Kategori();
            kat.setIdkategori(rs.getInt("idkategori"));
            kat.setNama(rs.getString("nama"));
            kat.setKeterangan(rs.getString("keterangan"));

            listKategori.add(kat);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return listKategori;
}
```

9. Tambahkan method **save()**. Method ini memiliki dua fungsi, yaitu insert dan update. Jika idkategori bernilai 0 maka data yang diinputkan belum ada di database sehingga akan diinsert. Sedangkan jika idkategori tidak bernilai 0 maka akan dilakukan proses update.

```
public void save(){
    if (this.idkategori == 0) {
        String query = "INSERT INTO kategori (nama, keterangan) VALUES ("
                + " '" + this.nama + "', "
                + " '" + this.keterangan + "') ";

        this.idkategori = DBHelper.insertQueryGetId(query);
    }
    else{
        String query = "UPDATE kategori SET "
                + " nama = '" + this.nama + "', "
                + " keterangan = '" + this.keterangan + "' "
                + " WHERE idkategori = '" + this.idkategori + "'";

        DBHelper.executeQuery(query);
    }
}
```

10. Tambahkan method **delete()** untuk melakukan operasi penghapusan pada tabel kategori pada database.

```
public void delete()
{
    String SQL = "DELETE FROM kategori WHERE idkategori = '" + this.idkategori + "'";
    DBHelper.executeQuery(SQL);
}
```

3.5 Percobaan 5

Mencoba backed yang sudah dibuat dengan mengoperasikannya lewat frontend berbasis teks (console). Percobaan ini dapat anda skip jika anda telah yakin bahwa backend yang anda buat sudah berfungsi dengan baik.

1. Pada package **frontend**, buat class **TestBackend**. Tambahkan import backend.*
2. Berikut kode lengkap untuk class TestBackend.

```
import backend.*;

public class TestBackend {
    public static void main(String[] args)
    {
        Kategori kat1 = new Kategori("Novel", "Koleksi buku novel");
        Kategori kat2 = new Kategori("Referensi", "Buku referensi ilmiah");
        Kategori kat3 = new Kategori("Komik", "Komik anak-anak");

        //test insert
        kat1.save();
        kat2.save();
        kat3.save();

        // test update
        kat2.setKeterangan("Koleksi buku referensi ilmiah");
        kat2.save();

        // test delete
        kat1.delete();

        // test select all
        for(Kategori k : Kategori.getAll())
        {
            System.out.println("Nama: " + k.getNama() + ", Ket: " + k.getKeterangan());
        }
```

```
        // test search
        for(Kategori k : Kategori.search("referensi"))
        {
            System.out.println("Nama: " + k.getNama() + ", Ket: " + k.getKeterangan());
        }
    }
}
```

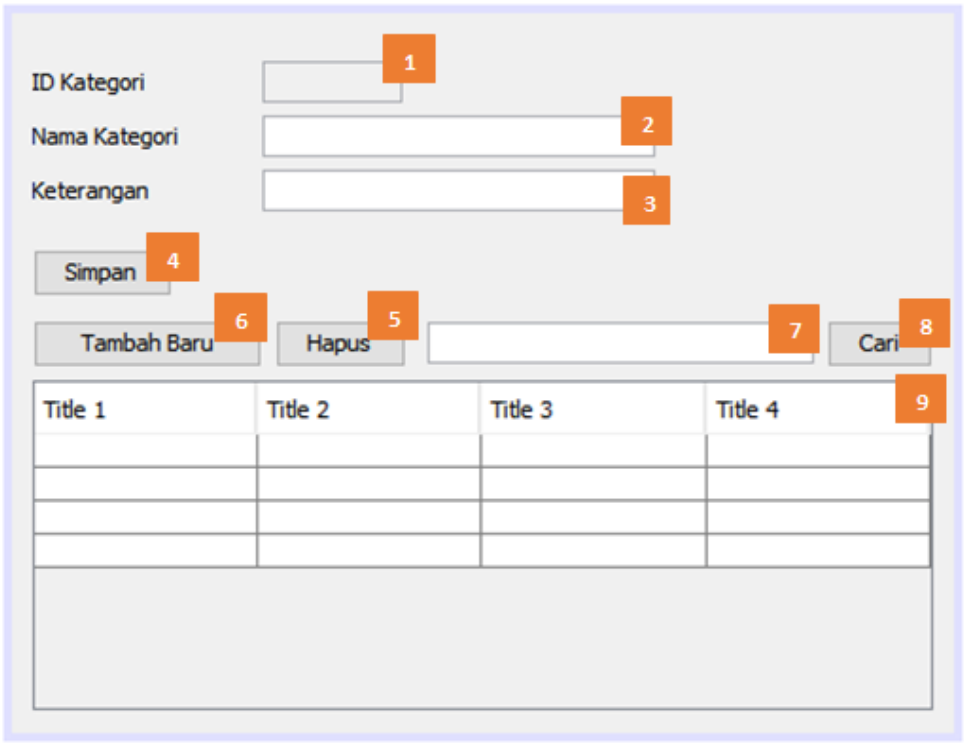
3. Jalankan TestBackend dengan klik kanan, Run File. Cocokkan outputnya:

```
run:
Nama: Referensi, Ket: Koleksi buku referensi ilmiah
Nama: Komik, Ket: Komik anak-anak
Nama: Referensi, Ket: Koleksi buku referensi ilmiah
BUILD SUCCESSFUL (total time: 1 second)
```

3.6 Percobaan 6

Pada percobaan ini kita akan membuat interface GUI untuk class **Kategori**.

1. Pada package **frontend**, buat JFrame dengan nama FrmKategori. Caranya adalah, klik kanan pada package frontend → New →JFrame Form.
2. Susun form sehingga seperti Gambar berikut dengan cara *drag & drop* komponennya dari "Palette" di sisi kanan atas.



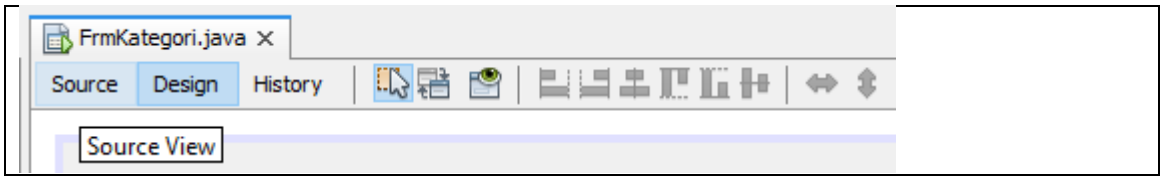
3. Perhatikan tabel di bawah ini kemudian atur properti setiap komponen dengan klik pada komponen, lalu ubah properties nya pada window **Properties** (umunya di sebelah kanan). Jika window belum terbuka, aktifkan melalui menu Window > IDE Tools > Properties. Properti *text* dan

enable ada pada tab **Properties**, sedangkan Variable Name ada pada tab **Code**. Variable Name juga dapat diset dengan klik kanan pada komponen kemudian pilih “Change Variable Name”

Tabel Pengaturan Properti

Nomor	Tipe Komponen	Variable Name	Properties
1	JTextField	txtIdKategori	text: kosong, enable: unchecked
2	JTextField	txtNama	text: kosong
3	JTextField	txtKeterangan	text: kosong
4	JButton	btnSimpan	text: Simpan
5	JButton	btnHapus	text: Hapus
6	JButton	btnTambahBaru	text: Tambah Baru
7	JTextField	txtCari	text: kosong
8	JButton	btnCari	text: Cari
9	JTable	tblKategori	

4. Edit kode program dengan klik Source tab



5. Tambahkan import backend.*, java.swing.JOptionPane, java.util.ArrayList, dan javax.swing.table.DefaultTableModel;

```
import backend.*;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

6. Tambahkan method **kosongkanForm()** untuk mengosongkan isian textbox pada form.

```
public void kosongkanForm()
{
    txtIdKategori.setText("0");
    txtNama.setText("");
    txtKeterangan.setText("");
}
```

7. Tambahkan method **tampilkanData()** untuk memperoleh semua data kategori dari database dengan method Kategori.getAll() kemudian menampilkannya ke JTable tblKategori.

   Keterangan:
   a. **Object**: class Object merupakan root class dari hierarki class. Setiap class merupakan turunan dari class Objek.
   b. **DefaultTableModel**: Class ini berfungsi untuk menyimpan nilai setiap cell yang akan ditampilkan dalam JTable. DefaultTableModel akan terdiri dari beberapa rowData. Setiap kolom dalam masing-masing rowData diisi dengan nilai dari listKategori yang diperoleh dari database.

Langkah:

a. Buat array of object baru bernama namaKolom untuk menyimpan nama-nama kolom
b. Buat objek bertipe DefaultTableModel baru bernama model yang akan menyimpan nilai setiap cell
c. Set namaKolom sebagai columnIdentifier(header) pada model
d. Dapatkan semua data kategori dari database menggunakan method getAll() kemudian simpan dalam listKategori
e. Buat array of object bernama rowData dengan jumlah elemen 3 sesuai kolom yang akan ditampilkan
f. Untuk setiap objek kat dalam listKategori, buat rowData dengan informasi id, nama, dan keterangan yang sesuai kemudian tambahkan ke model
g. Set model sebagai nilai yang akan ditampilkan oleh tblKategori

```
public void tampilkanData(){
    Object[] namaKolom = new Object[] {"ID", "Nama", "Kategori"};

    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(namaKolom);

    ArrayList<Kategori> listKategori = Kategori.getAll();
    Object[] rowData = new Object[3];

    for (Kategori kat : listKategori) {
        rowData[0] = kat.getIdkategori();
        rowData[1] = kat.getNama();
        rowData[2] = kat.getKeterangan();

        model.addRow(rowData);
    }

    tblKategori.setModel(model);
}
```

Judul kolom, sesuaikan dengan yang ada di database, tabel kategori

Jumlah kolom yang akan ditampilkan (3)

8. Tambahkan method **cari()** untuk melakukan pencarian berdasarkan keyword tertentu.

```
public void cari(String keyword){
    Object[] namaKolom = new Object[] {"ID", "Nama", "Kategori"};

    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(namaKolom);

    ArrayList<Kategori> listKategori = Kategori.search(keyword);
    Object[] rowData = new Object[3];

    for (Kategori kat : listKategori) {
        rowData[0] = kat.getIdkategori();
        rowData[1] = kat.getNama();
        rowData[2] = kat.getKeterangan();

        model.addRow(rowData);
    }

    tblKategori.setModel(model);
}
```

9. Pada konstruktor, tambahkan pemanggilan method kosongkanForm() dan tampilkanData(), agar ketika form ditampilkan pertama kali, maka form isian akan kosong dan list kategori langsung ditampilkan.

```
public FrmKategori()
{
    initComponents();
    tampilkanData();
    kosongkanForm();
}
```

10. Pada tab "Design", double klik pada **btnTambahBaru** untuk mengosongkan form sehingga dapat digunakan untuk menginputkan data baru. Method ini akan dipanggil setiap kali btnTambahBaru diklik.

```
private void btnTambahBaruActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    kosongkanForm();
}
```

11. Pada tab "Design", double klik pada **btnCari** untuk melakukan pencarian terhadap keyword yang dimasukkan pada **txtCari**. Method ini akan dipanggil setiap kali btnCari diklik.

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

12. Agar row data pada **tblKategori** dapat dipilih untuk diedit atau dihapus, maka perlu ditambahkan event mouse click pada **tblKategori**. Ketika pengguna mengklik pada tblKategori, maka data tersebut akan ditampilkan pada form di atas tabel. Caranya, klik kanan pada **tblKategori**, pilih Events → Mouse → MouseClicked. Tambahkan kode berikut ini:

```
private void tblKategoriMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    DefaultTableModel model = (DefaultTableModel)tblKategori.getModel();
    int row = tblKategori.getSelectedRow();

    txtIdKategori.setText(model.getValueAt(row, 0).toString());
    txtNama.setText(model.getValueAt(row, 1).toString());
    txtKeterangan.setText(model.getValueAt(row, 2).toString());
}
```

Kolom ke-0, utk mengeset txtIdKategori,

kolom ke-1 utk mengeset txtNama,

kolom ke-2 untuk mengeset txtKeterangan

Pengisian form secara lengkap dengan data yang berasal dari data table memungkinkan jika table menampilkan seluruh data. Jika tidak, tentukan idkategori kemudian dapatkan data lengkapnya menggunakan method getById().

13. Pada tab "Design", double click pada **btnSimpan**. Kita akan diarahkan ke method btnSimpanActionPerformed() yang akan dieksekusi ketika btnSimpan diklik.

    Langkah:

    a. Simpan data yang diinputkan user di text field ke variabel

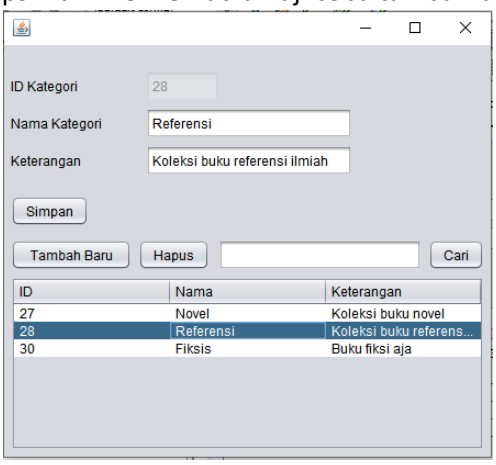
b. Cek apakah variabel untuk nama dan keterangan tidak kosong
c. Jika data nama dan keterangan kosong, munculkan warning dengan JOptionPane
d. Jika nama dan keterangan tidak kosong, dilakukan pembuatan objek kategori baru berdasarkan data tersebut.
e. Lakukan pemanggilan method save() untuk menyimpan data kategori ke database. Method save() sudah secara otomatis menentukan apakah data akan di-insert atau di-update.
f. Panggil method kosongkanForm() dan tampilkanData() untuk mengosongkan form dan memperbarui isi table

```
private void btnSimpanActionPerformed(java.awt.event.ActionEvent evt) {
    String nama = txtNama.getText();
    String keterangan = txtKeterangan.getText();

    if (!(nama.isEmpty() && keterangan.isEmpty())) {
        Kategori kat = new Kategori();
        kat.setIdkategori(Integer.parseInt(txtIdKategori.getText()));
        kat.setNama(nama);
        kat.setKeterangan(keterangan);
        kat.save();

        tampilkanData();
    }
    else{
        JOptionPane.showMessageDialog(null, "Silakan isi nama dan keterangan");
    }
}
```

g. Double klik pada **btnHapus** untuk menambahkan kode untuk menghapus data.

```
private void bthHapusActionPerformed(java.awt.event.ActionEvent evt) {
    Kategori kat = new Kategori();
    kat.setIdkategori(Integer.parseInt(txtIdKategori.getText()));
    kat.delete();

    kosongkanForm();
    tampilkanData();
}
```

h. Jalankan form dengan opsi Run File. Kemudian uji coba tambah baru, edit, hapus, cari.



3.7 Percobaan 7

Lakukan hal yang sama untuk data **Anggota**

1. Buat class **Anggota** pada package **backend**, lengkapi atribut dan method-nya.
2. Lakukan test pada class TestBackend pada package **frontend** (opsional)
3. Buat **FrmAnggota** pada package **frontend** dan lengkapi komponen, method, serta event-nya.

3.9 Percobaan 8

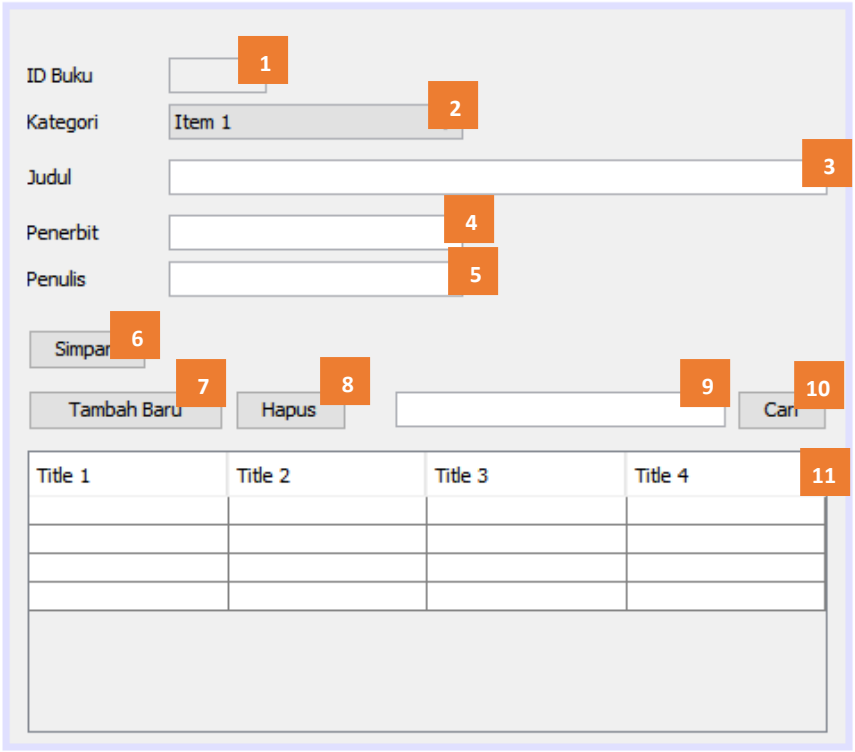
Untuk data **Buku**, caranya kurang lebih sama seperti data Kategori dan Anggota. Hanya saja yang berbeda adalah:

a. Pemanggilan **getKategori().getIdKategori()** pada query insert dan update untuk mengeset **idkategori** pada tabel **buku**
b. Query select yang melibatkan join table pada method getById(), getAll() dan search().

Kode lengkap class Buku dapat anda lihat di **Lampiran 1**. Untuk test buku pada **frontend**, bisa anda lihat di **Lampiran 2.**

Membuat GUI untuk data Buku, yang dilengkapi dengan combo box untuk memilih kategori yang terhubung dengan tabel kategori.

1. Pada package **frontend**, buat jFrame FrmBuku. Susun formnya sebagai berikut:



Nomor	Tipe Komponen	Variable Name	Properties
1	JTextField	txtIdBuku	text: kosong, enabled: uncheck
2	JComboBox	cmbKategori	
3	JTextField	txtJudul	text: kosong
4	JTextField	txtPenerbit	text: kosong
5	JTextField	txtPenulis	text: kosong
6	JButton	btnSimpan	text: Simpan
7	JButton	btnHapus	text: Hapus
8	JButton	btnTambahBaru	text: Tambah Baru
9	JTextField	txtCari	text: kosong
10	JButton	btnCari	text: Cari
11	JTable	tblBuku	

2. Tambahkan import backend.*, java.swing.JOptionPane, java.util.ArrayList, dan javax.swing.table.DefaultTableModel;

```
import backend.*;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

3. Tambahkan method **kosongkanForm()** untuk mengosongkan isian textbox pada form.

```
    public void kosongkanForm()
    {
        txtIdBuku.setText("0");
        cmbKategori.setSelectedIndex(0);
        txtJudul.setText("");
        txtPenulis.setText("");
        txtPenerbit.setText("");
    }
```

4. Tambahkan method **tampilkanData()** untuk mengambil semua data buku dari database dan menampilkannya ke JTable tblBuku.

```
public void tampilkanData(){
    Object[] namaKolom = new Object[] {"ID", "Kategori", "Judul", "Penerbit", "Penulis"};

    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(namaKolom);

    ArrayList<Buku> listBuku = Buku.getAll();
    Object[] rowData = new Object[5];

    for (Buku buku : listBuku) {
        rowData[0] = buku.getIdbuku();
        rowData[1] = buku.getKategori().getNama();
        rowData[2] = buku.getJudul();
        rowData[3] = buku.getPenerbit();
        rowData[4] = buku.getPenulis();

        model.addRow(rowData);
    }

    tblBuku.setModel(model);
}
```

5. Tambahkan method **cari()** untuk melakukan pencarian berdasarkan keyword tertentu.

```
public void cari(String keyword){
    Object[] namaKolom = new Object[] {"ID", "Kategori", "Judul", "Penerbit", "Penulis"};

    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(namaKolom);

    ArrayList<Buku> listBuku = Buku.search(keyword);
    Object[] rowData = new Object[5];

    for (Buku buku : listBuku) {
        rowData[0] = buku.getIdbuku();
        rowData[1] = buku.getKategori().getNama();
        rowData[2] = buku.getJudul();
        rowData[3] = buku.getPenerbit();
        rowData[4] = buku.getPenulis();

        model.addRow(rowData);
    }

    tblBuku.setModel(model);
}
```

6. Tambahkan method **setCmbKategori().** Pada method ini semua kategori didapatkan dari database menggunakan method Kategori.getAll() kemudian dijadikan sebagai opsi untuk cmbKategori.

```
public void setCmbKategori(){
    DefaultComboBoxModel model = new DefaultComboBoxModel(Kategori.getAll().toArray());
    cmbKategori.setModel(model);
}
```

7. Agar cmbKategori menampilkan **nama kategori**, maka override method **toString()** pada class **Kategori.** Tambahkan kode berikut ini pada class **Kategori**:

```
public String toString()
{
    return nama;
}
```

8. Pada konstruktor, tambahkan pemanggilan method kosongkanForm(), tampilkanCmbKategori() dan tampilkanData(), agar ketika form ditampilkan pertama kali, maka form isian akan kosong dan data buku langsung ditampilkan, dan opsi untuk kategori sudah disediakan.

```
public FrmBuku() {
    initComponents();
    tampilkanData();
    tampilkanCmbKategori();
    kosongkanForm();
}
```

9. Double klik pada **btnTambahBaru** untuk mengosongkan form sehingga dapat digunakan untuk menginputkan data baru.

```
private void btnTambahBaruActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    kosongkanForm();
}
```

10. Double klik pada **btnCari** untuk melakukan pencarian terhadap keyword yang dimasukkan pada **txtCari**.

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

11. Agar row data pada **tblBuku** dapat dipilih untuk diedit atau dihapus, maka perlu ditambahkan event mouse click pada tblBuku. Ketika pengguna mengklik pada tblBuku, maka data tersebut akan ditampilkan pada form di atas tabel. Caranya, klik kanan pada tblBuku, pilih Events → Mouse → MouseClicked.

    Pada kode program event mouse click nya **tblKategori**, nilai pada text field diset berdasarkan data pada tabel. Namun untuk **tblBuku**, form memerlukan data idkategori yang tidak tersedia pada tabel. Oleh karena itu, langkah yang dilakukan adalah:

    a. Tentukan row yang diklik
    b. Tentukan idbuku
    c. Tentukan objek buku yang sesuai dengan menggunakan method getById()
    d. Isi form sesuai data buku

```
private void tblBukuMouseClicked(java.awt.event.MouseEvent evt) {
    DefaultTableModel model = (DefaultTableModel)tblBuku.getModel();
    int rowIndex = tblBuku.getSelectedRow();
    int idbuku = Integer.parseInt(model.getValueAt(rowIndex, 0).toString());

    Buku buku = Buku.getById(idbuku);

    txtIdBuku.setText(Integer.toString(idbuku));
    cmbKategori.getModel().setSelectedItem(buku.getKategori());
    txtJudul.setText(buku.getJudul());
    txtPenerbit.setText(buku.getPenerbit());
    txtPenulis.setText(buku.getPenulis());
}
```

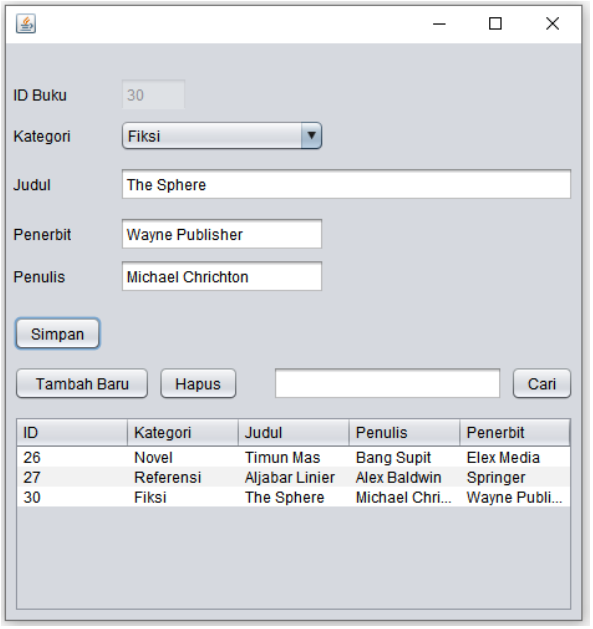
12. Double klik pada **btnHapus** untuk menambahkan kode untuk menghapus data.

```
private void bthHapusActionPerformed(java.awt.event.ActionEvent evt) {
    Buku buku = new Buku();
    buku.setIdbuku(Integer.parseInt(txtIdBuku.getText()));
    buku.delete();

    kosongkanForm();
    tampilkanData();
}
```

13. Double klik pada **btnSimpan** untuk menambahkan kode untuk menyimpan data. Method save() pada class Buku sudah secara otomatis menentukan apakah data akan di-insert atau di-update.

```
private void btnSimpanActionPerformed(java.awt.event.ActionEvent evt) {
    String judul = txtJudul.getText();
    String penerbit = txtPenerbit.getText();
    String penulis = txtPenulis.getText();

    if (!(judul.isEmpty() && penerbit.isEmpty() && penulis.isEmpty())) {
        Buku buku = new Buku();
        buku.setIdbuku(Integer.parseInt(txtIdBuku.getText()));
        buku.setKategori((Kategori)cmbKategori.getSelectedItem());
        buku.setJudul(judul);
        buku.setPenerbit(penerbit);
        buku.setPenulis(penulis);
        buku.save();

        tampilkanData();
    }
    else{
        JOptionPane.showMessageDialog(null, "Silakan isi form dengan lengkap");
    }
}
```

14. Jalankan form dengan opsi Run File. Kemudian ujicoba tambah baru, edit, hapus, cari.



15. Update method **delete()** pada class **Kategori** yang mencegah penghapusan kategori jika ada buku yang tersimpan pada kategori tersebut. Method ini akan mengembalika nilai true jika tidak ada buku terkait dan kategori berhasil dihapus. Jika tidak maka akan mengembalikan nilai false.

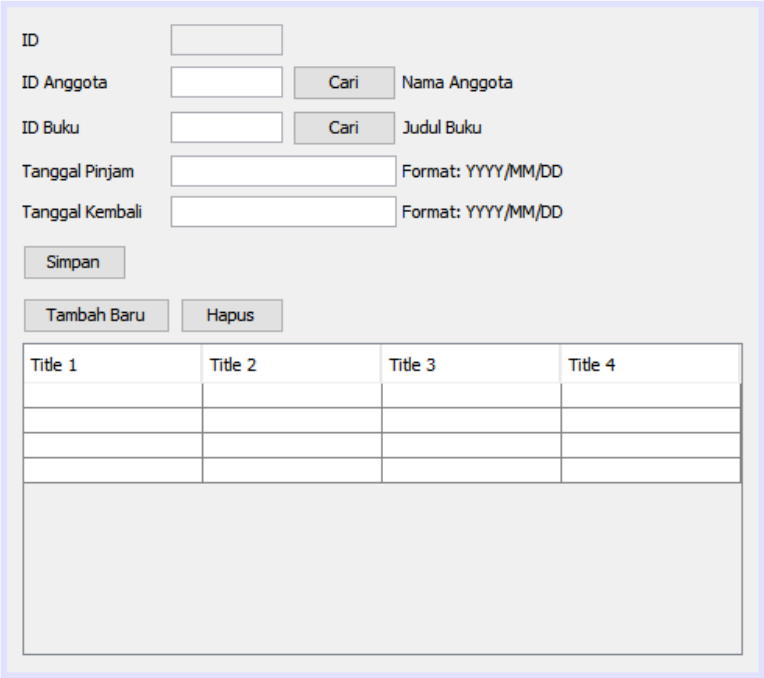
```
public boolean delete(){
    ResultSet rs = DBHelper.selectQuery("SELECT COUNT(*) as jumlahBuku FROM buku"
                                        + " WHERE idkategori = " + this.idkategori + "");
    int jumlahBuku = 0;

    try {
        while (rs.next()) {
            jumlahBuku = rs.getInt("jumlahBuku");
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    if (jumlahBuku == 0) {
        String query = "DELETE FROM kategori WHERE idkategori = " + this.idkategori;
        DBHelper.executeQuery(query);
        return true;
    }
    else{
        return false;
    }
}
```

16. Update class FrmKategori dengan langkah sebagai berikut:
    a. Dapatkan nilai idkategori yang akan dihapus dari text field txtidkategori
    b. Panggil method delete() untuk objek kat yang dibuat dengan id tersebut
    c. Jika penghapusan data berhasil dilakukan, kosongkan form lalu perbaharui tblKategori
    d. Jika penghapusan data tidak berhasil karena ada buku pada kategori tersebut, munculkan warning dengan JOptionPane

```
private void bthHapusActionPerformed(java.awt.event.ActionEvent evt) {
    Kategori kat = new Kategori();
    kat.setIdkategori(Integer.parseInt(txtIdKategori.getText()));
    boolean deleteSuccess = kat.delete();

    if (deleteSuccess) {
        kosongkanForm();
        tampilkanData();
    }
    else{
        JOptionPane.showMessageDialog(null, "Terdapat data buku pada kategori tersebut");
    }

}
```

4. Tugas

1. Buatlah class **Peminjaman.**
2. Buatlah form **FrmPeminjaman** dan susun sebagai berikut:



3. Atur kode program agar dapat menangani transaksi peminjaman dan pengembalian.

   Note:
   Pada textbox ID Anggota, pengguna tinggal memasukkan ID anggota, kemudian menekan tombol Cari. Jika ketemu, maka label **"Nama Anggota"** yang ada di samping tombol Cari tersebut akan menampilkan nama anggota dari ID yang dimasukkan tadi. Begitu juga dengan ID Buku.

Lampiran

Lampiran 1. Kode lengkap class Buku

```
package backend;
import java.util.ArrayList;
import java.sql.*;

public class Buku {
    private int idbuku;
    private Kategori kategori;
    private String judul;
    private String penerbit;
    private String penulis;

    public Buku() {
    }

    public Buku(Kategori kategori, String judul, String penerbit, String penulis) {
        this.kategori = kategori;
        this.judul = judul;
        this.penerbit = penerbit;
        this.penulis = penulis;
    }

    public int getIdbuku() {
        return idbuku;
    }

    public void setIdbuku(int idbuku) {
        this.idbuku = idbuku;
    }

    public String getJudul() {
        return judul;
    }

    public void setJudul(String judul) {
        this.judul = judul;
    }

    public Kategori getKategori() {
        return kategori;
    }

    public void setKategori(Kategori kategori) {
        this.kategori = kategori;
    }

    public String getPenerbit() {
        return penerbit;
    }

    public void setPenerbit(String penerbit) {
        this.penerbit = penerbit;
    }

    public String getPenulis() {
        return penulis;
    }

    public void setPenulis(String penulis) {
        this.penulis = penulis;
    }
```

```
public static ArrayList<Buku> getAll(){
    ArrayList<Buku> listBuku = new ArrayList();

    String query = "SELECT buku.*, kategori.nama, kategori.keterangan"
                 + " FROM buku"
                 + " LEFT JOIN kategori ON buku.idkategori = kategori.idkategori";

    ResultSet rs = DBHelper.selectQuery(query);

    try {
        while (rs.next()) {
            Kategori kat = new Kategori();
            kat.setIdkategori(rs.getInt("idkategori"));
            kat.setNama(rs.getString("nama"));
            kat.setKeterangan(rs.getString("keterangan"));

            Buku buku = new Buku();
            buku.setIdbuku(rs.getInt("idbuku"));
            buku.setKategori(kat);
            buku.setJudul(rs.getString("judul"));
            buku.setPenerbit(rs.getString("penerbit"));
            buku.setPenulis(rs.getString("penulis"));

            listBuku.add(buku);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return listBuku;
}
```

```
public static Buku getById(int id){
    Buku buku = null;

    String query = "SELECT buku.*, kategori.nama, kategori.keterangan"
                 + " FROM buku"
                 + " LEFT JOIN kategori ON buku.idkategori = kategori.idkategori"
                 + " WHERE idbuku = " + id;

    ResultSet rs = DBHelper.selectQuery(query);

    try {
        while (rs.next()) {
            Kategori kat = new Kategori();
            kat.setIdkategori(rs.getInt("idkategori"));
            kat.setNama(rs.getString("nama"));
            kat.setKeterangan(rs.getString("keterangan"));

            buku = new Buku();
            buku.setIdbuku(rs.getInt("idbuku"));
            buku.setKategori(kat);
            buku.setJudul(rs.getString("judul"));
            buku.setPenerbit(rs.getString("penerbit"));
            buku.setPenulis(rs.getString("penulis"));
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return buku;
}
```

```
    public static ArrayList<Buku> search(String keyword){
        ArrayList<Buku> listBuku = new ArrayList();

        String query = "SELECT buku.*, kategori.nama, kategori.keterangan"
                     + " FROM buku"
                     + " LEFT JOIN kategori ON buku.idkategori = kategori.idkategori"
                     + " WHERE judul LIKE '%" + keyword + "%'"
                     + " OR penerbit LIKE '%" + keyword + "%'"
                     + " OR penulis LIKE '%" + keyword + "%'";

        ResultSet rs = DBHelper.selectQuery(query);

        try {
            while (rs.next()) {
                Kategori kat = new Kategori();
                kat.setIdkategori(rs.getInt("idkategori"));
                kat.setNama(rs.getString("nama"));
                kat.setKeterangan(rs.getString("keterangan"));

                Buku buku = new Buku();
                buku.setIdbuku(rs.getInt("idbuku"));
                buku.setKategori(kat);
                buku.setJudul(rs.getString("judul"));
                buku.setPenerbit(rs.getString("penerbit"));
                buku.setPenulis(rs.getString("penulis"));

                listBuku.add(buku);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        return listBuku;
    }
```

```
    public void save(){
        if (this.idbuku == 0) {
            String query = "INSERT INTO buku (idkategori, judul, penerbit, penulis) VALUES ("
                         + " " + this.kategori.getIdkategori() + ", "
                         + " '" + this.judul + "', "
                         + " '" + this.penerbit + "', "
                         + " '" + this.penulis + "') ";

            this.idbuku = DBHelper.insertQueryGetId(query);
        }
        else{
            String query = "UPDATE buku SET "
                         + " idkategori = " + this.kategori.getIdkategori() + ", "
                         + " judul = '" + this.judul + "', "
                         + " penerbit = '" + this.penerbit + "', "
                         + " penulis = '" + this.penulis + "' "
                         + " WHERE idbuku = '" + this.idbuku + "'";

            DBHelper.executeQuery(query);
        }
    }

    public void delete(){
        String query = "DELETE FROM buku WHERE idbuku = " + this.idbuku;
        DBHelper.executeQuery(query);
    }
}
```

Lampiran 2. Kode untuk TestBackend

```
import backend.*;

public class TestBackend {
    public static void main(String[] args)
    {
        Kategori novel = new Kategori("Novel", "Koleksi buku novel");
        Kategori referensi = new Kategori("Referensi", "Buku referensi ilmiah");

        //test getById
        Kategori kat1 = Kategori.getById(25);

        if (kat1 != null) {
            System.out.println(kat1.getNama());
        }

        novel.save();
        referensi.save();

        Buku buku1 = new Buku(novel, "Timun Mas", "Elex Media", "Bang Supit");
        Buku buku2 = new Buku(referensi, "Metode Linier", "Springer", "Alex
Baldwin");
        Buku buku3 = new Buku(novel, "Bintang Terang", "Erlangga", "Mat Sewoot");

        // test insert
        buku1.save();
        buku2.save();
        buku3.save();

        // test update
        buku2.setJudul("Aljabar Linier");
        buku2.save();

        //test getById
        Buku buku = Buku.getById(1);

        if (buku != null) {
            System.out.println("Kategori: " + buku.getKategori().getNama() + ",
Judul: " + buku.getJudul());
        }

        // test delete
        buku3.delete();

        // test select all
        for(Buku b : Buku.getAll())
        {
            System.out.println("Kategori: " + b.getKategori().getNama() + ",
Judul: " + b.getJudul());
        }

        // test search
        for(Buku b : Buku.search("timun"))
        {
            System.out.println("Kategori: " + b.getKategori().getNama() + ",
Judul: " + b.getJudul());
        }
    }
}
```

Lampiran 3. Static keyword

Static attribute → attribute yang hanya terkait pada class, tetapi tidak terkait pada suatu objek tertentu
Static method → method yang hanya terkait pada class, tetapi tidak terkait pada suatu objek tertentu

Method getJudul() bukan merupakan method statis, artinya method ini terkait pada objek.

- buku1.getJudul() → mengembalikan nilai atribut judul dari buku1
- buku2.getJudul() → mengembalikan nilai atribut judul dari buku2
- buku3.getJudul() → mengembalikan nilai atribut judul dari buku3
- dst

Terkadang kita ingin membuat method yang terkait pada class Buku, tetapi tidak ada kaitannya dengan objek buku tertentu. Misal method getAll() pada class Buku berfungsi untuk mengembalikan semua data buku dari database. Jika method getAll() bukan static method, pemanggilan method harus melalui objek seperti biasanya, misalnya:

- buku4.getAll() → mengembalikan semua data buku dari database
- buku5.getAll() → mengembalikan semua data buku dari database
- buku6.getAll() → mengembalikan semua data buku dari database
- dst

Ketiga method tersebut melakukan hal yang persis sama dan tidak bergantung pada objek yang memanggilnya. Jika suatu method bersifat demikian, maka method tersebut sebaiknya dibuat sebagai static method sehingga dapat diakses langsung menggunakan nama class nya tanpa membuat objek terlebih dahulu.

- Buku.getAll()
- Buku.search(String keyword)
- Buku.getById(int id)

CATATAN:

Method delete() pada contoh kasus Jobsheet saat ini bukan static method. Jadi untuk memanggil method delete() harus menggunakan suatu objek:

```
public void delete(){
    String query = "DELETE FROM buku WHERE idbuku = " + this.idbuku;
    DBHelper.executeQuery(query);
}
```

```
private void bthHapusActionPerformed(java.awt.event.ActionEvent evt) {
    Buku buku = new Buku();
    buku.setIdbuku(Integer.parseInt(txtIdBuku.getText()));
    buku.delete();

    kosongkanForm();
    tampilkanData();
}
```

Tapi sebenarnya method delete juga bisa dibuat sebagai static method delete(int id) yang memungkinkan penghapusan objek buku berdasarkan id tanpa harus dibuat objeknya terlebih dahulu.

Static method Delete:

```
public static void delete(int id){
    String query = "DELETE FROM buku WHERE idbuku = " + id;
    DBHelper.executeQuery(query);
}
```

Pemanggilan static method Delete:

```
private void bthHapusActionPerformed(java.awt.event.ActionEvent evt) {
    int id = Integer.parseInt(txtIdBuku.getText());
    Buku.delete(id);

    kosongkanForm();
    tampilkanData();
}
```

Pada beberapa konsep pemrograman, method-method helper yang berfungsi untuk melakukan manipulasi database, yaitu method untuk Create Read Update Delete, umumnya dipisah dari class Buku menjadi class tersendiri, misalnya BukuHelper