ACTIVITY PERTEMUAN 4

Nama : Ferdy Agustian Prasetyo

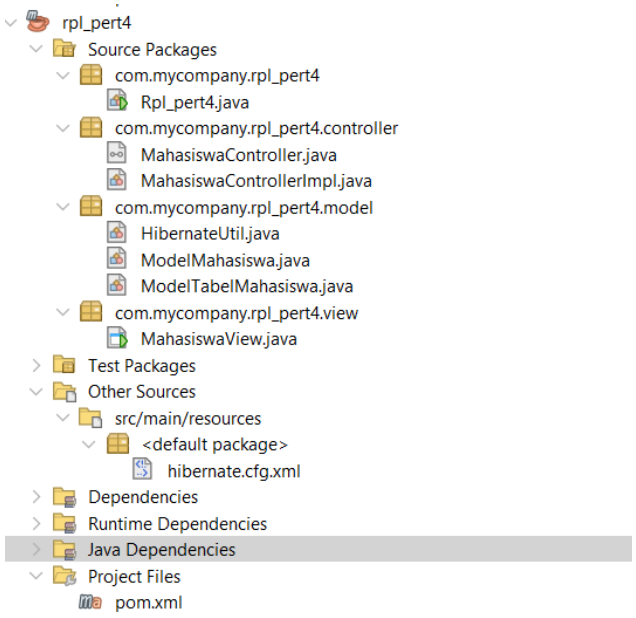
NPM 50422565

Kelas : 4IA28

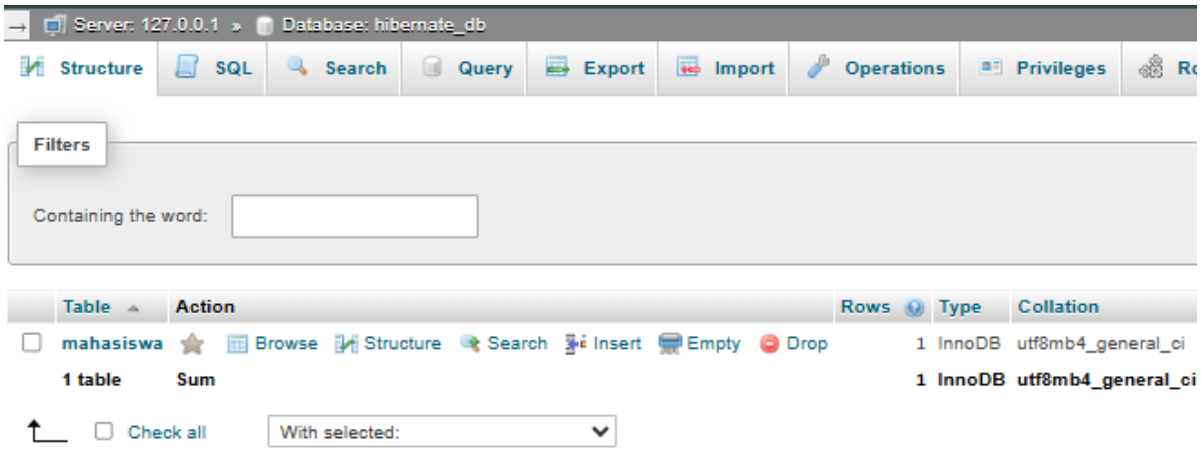
Materi : Object Relational Mapping

Mata Praktikum : Rekayasa Perangkat Lunak 2

Struktur Folder Program



Database



MahasiswaController.java

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.rpl_pert4.controller;

import com.mycompany.rpl_pert4.model.ModelMahasiswa;
import java.util.List;

/**
 *
 * @author ASUS
 */
public interface MahasiswaController {
    public void addMhs(ModelMahasiswa mhs);
    public ModelMahasiswa getMhs(int id);
    public void updateMhs(ModelMahasiswa mhs);
    public void deleteMhs(int id);
    public List<ModelMahasiswa> getAllMahasiswa();
}
```

Logika dari kode yang ditampilkan tersebut bukanlah logika eksekusi, melainkan sebuah logika desain arsitektural. Kode itu mendefinisikan sebuah interface publik bernama MahasiswaController, yang berperan sebagai "kontrak" atau *blueprint* formal untuk lapisan *controller* dalam sebuah pola desain MVC. Interface ini tidak memiliki implementasi (logika "bagaimana"), tetapi *merinci* dan *memaksa* kelas konkret mana pun (misalnya MahasiswaControllerImpl) yang mengimplementasikannya untuk wajib menyediakan lima *method* fungsional yang terkait operasi CRUD (Create, Read, Update, Delete): addMhs (untuk menambah data), getMhs (untuk mengambil satu data berdasarkan id), updateMhs (untuk memodifikasi data), deleteMhs (untuk menghapus data berdasarkan id), dan getAllMahasiswa (untuk mengambil semua data). Tujuan utama dari desain ini adalah untuk mencapai pemisahan tugas (*separation of concerns*), memungkinkan View (antarmuka pengguna) berinteraksi dengan kontrak ini tanpa perlu mengetahui detail implementasi di *backend* (apakah menggunakan JDBC, Hibernate, atau lainnya), sehingga meningkatkan fleksibilitas dan kemudahan perawatan kode.

MahasiswaControllerMPL.java

```java
package com.mycompany.rpl_pert4.controller;

import com.mycompany.rpl_pert4.model.HibernateUtil;
import com.mycompany.rpl_pert4.model.ModelMahasiswa;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

/**
 *
 * @author ASUS
 */
public class MahasiswaControllerImpl implements MahasiswaController{

    @Override
    public void addMhs(ModelMahasiswa mhs){
        Transaction trx = null;

        try (Session session = HibernateUtil.getSessionFactory().openSession()){
            trx = session.beginTransaction();
            session.save(mhs);
            trx.commit();
        }catch (Exception e){
            if (trx != null){
                trx.rollback();
            }
            e.printStackTrace();
        }
    }
```

```java
@Override
public void updateMhs(ModelMahasiswa mhs) {
    Transaction trx = null;

    try (Session session = HibernateUtil.getSessionFactory().openSession()){
        trx = session.beginTransaction();
        session.update(mhs);
        trx.commit();
    } catch (Exception e){
        if (trx != null){
            trx.rollback();
        }
        e.printStackTrace();
    }

}
```

```java
@Override
public void deleteMhs(int id) {
    Transaction trx = null;

    try (Session session = HibernateUtil.getSessionFactory().openSession()){
        trx = session.beginTransaction();
        ModelMahasiswa mhs = session.get(ModelMahasiswa.class, id);
        if(mhs != null){
            session.delete(mhs);
            System.out.println("Berhasil hapus");
        }
        trx.commit();
    } catch (Exception e){
        if (trx != null){
            trx.rollback();
        }
        e.printStackTrace();
    }

}
```

```java
@Override
public List<ModelMahasiswa> getAllMahasiswa() {
    Transaction trx = null;
    List<ModelMahasiswa> listMhs = null;

    try (Session session = HibernateUtil.getSessionFactory().openSession()){
        trx = session.beginTransaction();
        // Using HQL (Hibernate Query Language) to fetch all records
        Query<ModelMahasiswa> query = session.createQuery("from ModelMahasiswa", ModelMahasiswa.class);
        listMhs = query.list(); // Fetch all results

        trx.commit(); // Commit transaction
    } catch (Exception e) {
        if (trx != null) {
            trx.rollback(); // Rollback transaction in case of error
        }
        e.printStackTrace();
    }

    // Return the fetched list
    return listMhs;
}

@Override
public ModelMahasiswa getMhs(int id) {
    throw new UnsupportedOperationException("Not supported yet.");
}

}
```

Logika program ini adalah implementasi konkret dari interface MahasiswaController yang menggunakan *framework* Hibernate untuk menjembatani antara objek Java (ModelMahasiswa) dan *database* relasional. Kelas MahasiswaControllerImpl ini menyediakan logika *backend* untuk operasi CRUD dengan mengikuti pola transaksi Hibernate yang aman: untuk setiap operasi yang mengubah data (addMhs, updateMhs, deleteMhs), kode ini membuka Session dari HibernateUtil, memulai Transaction, dan membungkus logika *database* dalam blok try-catch. Secara spesifik, addMhs menggunakan session.save(mhs) untuk mengeksekusi perintah INSERT, updateMhs menggunakan session.update(mhs) untuk UPDATE, dan deleteMhs pertama-tama mengambil objek dengan session.get(id) sebelum menghapusnya dengan session.delete(mhs). Jika operasi berhasil, transaksi akan di-commit; jika terjadi *error*, transaksi akan di-rollback untuk membatalkan perubahan dan menjaga integritas data. Untuk membaca data, getAllMahasiswa menggunakan HQL (Hibernate Query Language) melalui kueri "from ModelMahasiswa" untuk mengambil semua data dan memetakannya secara otomatis ke List<ModelMahasiswa>. Terakhir, *method* getMhs(int id) (untuk mengambil satu mahasiswa) sengaja dibiarkan tidak diimplementasikan, yang ditandai dengan pelemparan UnsupportedOperationException.

HibernateUtil.java

```
/*
    public ModelMahasiswa(int id, String npm, String nama, int semester, float ipk) {
        this.id = id;
        this.npm = npm;
        this.nama = nama;
        this.semester = semester;
        this.ipk = ipk;
    }
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNpm() {
        return npm;
    }

    public void setNpm(String npm) {
        this.npm = npm;
    }

    public String getNama() {
        return nama;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public int getSemester() {
        return semester;
    }

    public void setSemester(int semester) {
        this.semester = semester;
    }

    public float getIpk() {
        return ipk;
    }

    public void setIpk(float ipk) {
        this.ipk = ipk;
    }
```

Logika dari kelas HibernateUtil ini adalah untuk mengimplementasikan Singleton Pattern secara efisien bagi objek SessionFactory Hibernate, yang merupakan praktik standar karena SessionFactory adalah objek yang sangat "mahal" (memakan banyak sumber daya) untuk dibuat dan dirancang untuk bersifat *thread-safe* serta digunakan bersama oleh seluruh aplikasi. Ini dicapai dengan menggunakan blok inisialisasi statis (static { ... }), sebuah blok kode yang dijamin oleh JVM (Java Virtual Machine) hanya akan dieksekusi *satu kali*, yaitu saat kelas HibernateUtil pertama kali dimuat ke dalam memori. Di dalam blok ini, baris new Configuration().configure().buildSessionFactory() dieksekusi untuk membaca file konfigurasi *default* (biasanya hibernate.cfg.xml), memuat semua *mapping* entitas, dan membangun satu-satunya *instance* SessionFactory. Jika proses krusial ini gagal (misalnya, *database* tidak terjangkau atau konfigurasi salah), blok catch akan menangkap Throwable, mencatat *error* tersebut, dan kemudian melempar ExceptionInInitializerError—sebuah *error* fatal yang secara sengaja akan menghentikan aplikasi (prinsip *fail-fast*) karena aplikasi tidak dapat berfungsi tanpanya. Kelas ini kemudian menyediakan *method* publik public static SessionFactory getSessionFactory() yang berfungsi sebagai satu-satunya titik akses global bagi komponen lain (seperti MahasiswaControllerImpl) untuk mendapatkan *instance* SessionFactory yang tunggal tersebut. Terakhir, *method* testConnection disertakan sebagai utilitas diagnostik sederhana yang mencoba membuka dan langsung menutup Session (menggunakan try-with-resources) hanya untuk memverifikasi bahwa SessionFactory berhasil dibuat dan koneksi *database* dapat terjalin.

ModelMahasiswa.java

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.rpl_pert4.model;

import jakarta.persistence.*;

/**
 *
 * @author ASUS
 */
@Entity
@Table(name= "mahasiswa")
public class ModelMahasiswa {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="npm", nullable=false, length=10)
    private String npm;

    @Column(name="nama", nullable=false, length=50)
    private String nama;

    @Column(name="semester", nullable=false)
    private int semester;

    @Column(name="ipk", nullable=false)
    private float ipk;

    public ModelMahasiswa() {

    }
```

```java
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        ModelMahasiswa mahasiswa = mahasiswaList.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return mahasiswa.getId();
            case 1:
                return mahasiswa.getNpm();
            case 2:
                return mahasiswa.getNama();
            case 3:
                return mahasiswa.getSemester();
            case 4:
                return mahasiswa.getIpk();
            default:
                return null;
        }
    }

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

    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return false; // Semua sel tidak dapat diedit
    }

    // Method untuk menambahkan atau memodifikasi data, jika dibutuhkan
    public void setMahasiswaList(List<ModelMahasiswa> mahasiswaList) {
        this.mahasiswaList = mahasiswaList;
        fireTableDataChanged(); // Memberitahu JTable bahwa data telah berubah
    }


}
```

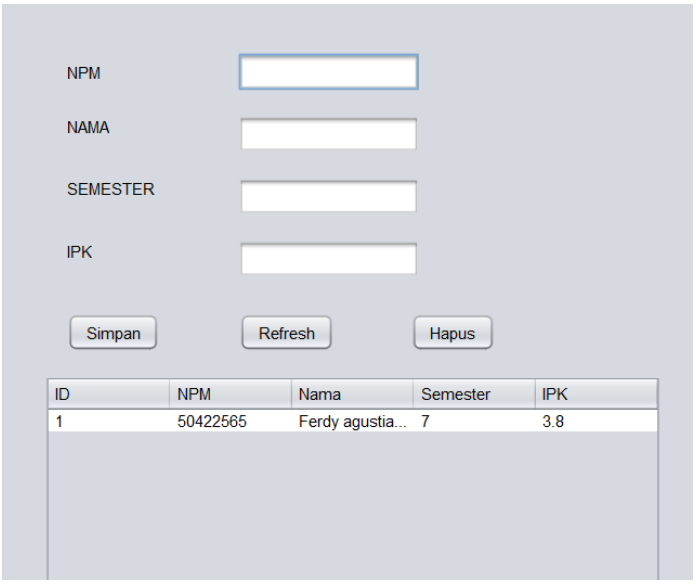
Logika dari kode ini adalah untuk mendefinisikan sebuah POJO (Plain Old Java Object), ModelMahasiswa, sebagai entitas persistensi yang diatur oleh JPA (Jakarta Persistence API), yang implementasinya (dalam kasus ini, Hibernate) akan mengelolanya. Ini dicapai dengan menggunakan *metadata* dalam bentuk anotasi: @Entity menandai kelas ini sebagai komponen yang akan dipetakan oleh ORM, sementara @Table(name="mahasiswa") secara eksplisit menautkan kelas ini ke tabel *database* yang bernama "mahasiswa". *Field* id ditetapkan sebagai *primary key* unik melalui @Id, dan anotasi @GeneratedValue(strategy = GenerationType.IDENTITY) menginstruksikan *provider* persistensi (Hibernate) untuk mendelegasikan pembuatan nilai *key* ini ke mekanisme *auto-increment* internal *database* (misalnya AUTO_INCREMENT di MySQL) setiap kali ada INSERT baru. *Field-field* lainnya (npm, nama, semester, ipk) dipetakan ke kolom tabel yang sesuai menggunakan @Column, yang juga mendefinisikan *constraints* level *database* (DDL) seperti nullable=false (menjadi NOT NULL) dan length (yang menentukan ukuran VARCHAR). Ketersediaan dua konstruktor—satu konstruktor default (tanpa argumen) yang wajib ada agar JPA/Hibernate dapat membuat *instance* objek saat mengambil data dari *database*, dan satu konstruktor *all-args* untuk kemudahan pembuatan objek di sisi aplikasi—serta kepatuhan penuh pada konvensi JavaBeans (menyediakan *method getter* dan *setter* publik untuk semua *field* privat) sangat krusial, karena inilah mekanisme yang digunakan *framework* untuk membaca dan menyuntikkan data ke dalam objek entitas.

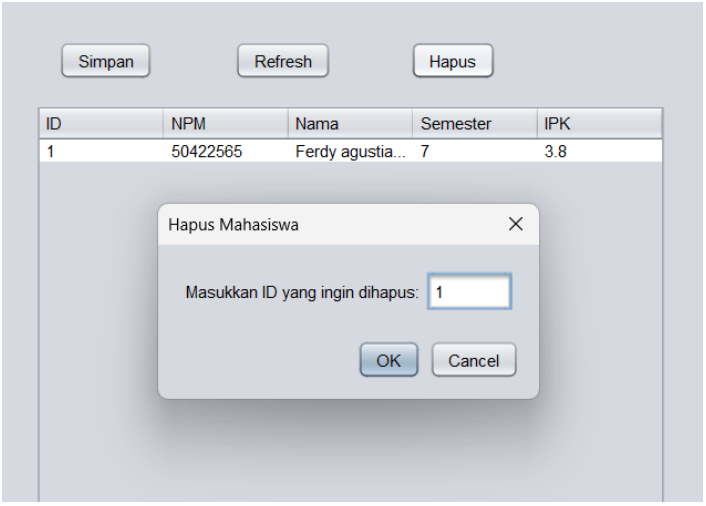
ModelTabelMahasiswa.java

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.rpl_pert4.model;

import java.util.List;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author ASUS
 */
public class ModelTabelMahasiswa extends AbstractTableModel {

    private List<ModelMahasiswa> mahasiswaList;
    private String[] columnNames = {"ID", "NPM", "Nama", "Semester", "IPK"};

    public ModelTabelMahasiswa(List<ModelMahasiswa> mahasiswaList) {
        this.mahasiswaList = mahasiswaList;
    }
    @Override
    public int getRowCount() {
        return mahasiswaList.size(); // Jumlah baris sesuai dengan jumlah data mahasiswa
    }

    @Override
    public int getColumnCount() {
        return columnNames.length; // Jumlah kolom sesuai dengan jumlah elemen dalam columnNames
    }
```

Logika dari kelas ModelTabelMahasiswa ini adalah untuk bertindak sebagai adaptor atau jembatan (bagian dari *design pattern* Adapter) yang menghubungkan sumber data (berupa List<ModelMahasiswa>) dengan komponen visual JTable dalam GUI Swing. Kelas ini meng-extends AbstractTableModel, yang memaksanya untuk menyediakan implementasi konkret atas *method-method* yang akan dipanggil oleh JTable untuk menggambar dirinya sendiri: getRowCount diimplementasikan untuk mengembalikan jumlah baris berdasarkan ukuran (size) dari mahasiswaList, sementara getColumnCount mengembalikan jumlah kolom berdasarkan panjang *array* columnNames. Logika intinya berada di dalam getValueAt, yang dipanggil oleh JTable untuk setiap sel; *method* ini terlebih dahulu mengambil objek ModelMahasiswa yang spesifik berdasarkan rowIndex, kemudian menggunakan switch *statement* pada columnIndex untuk memetakan dan mengembalikan nilai properti yang sesuai (seperti getId(), getNama(), atau getIpk()) untuk kolom tersebut. Selain itu, getColumnName digunakan untuk mengisi *header* tabel, isCellEditable diatur ke false untuk membuat tabel bersifat *read-only*, dan *method* setMahasiswaList diekspos agar *controller* dapat menyuntikkan data baru (misalnya setelah *refresh*), yang kemudian memanggil fireTableDataChanged() untuk memberi tahu JTable agar memperbarui tampilannya secara visual.

Output


NPM
NAMA
SEMESTER
IPK
Simpan
Refresh
Hapus
ID
NPM
Nama
Semester
IPK
1
50422565
Ferdy agustia...
7
3.8


Simpan
Refresh
Hapus
ID
NPM
Nama
Semester
IPK
1
50422565
Ferdy agustia...
7
3.8
Hapus Mahasiswa
Masukkan ID yang ingin dihapus:
1
OK
Cancel


Simpan
Refresh
Hapus
ID
NPM
Nama
Semester
IPK