LAPORAN AKHIR PRAKTIKUM

Mata Praktikum	: Rekayasa Perangkat Lunak 2
Kelas	: 4IA28
Praktikum ke	: 1
Tanggal	: 19 Oktober 2024
Materi	: Perkenalan OOP
NPM	: 50421118
Nama	: Alghany Sihombing
Ketua asisten	: Iqbal Al Fakhri
Nama asisten	:
Paraf asisten	:
Jumlah lembar	: 10 lembar

LABORATORIUM TEKNIK
INFORMATIKA UNIVERSITAS
GUNADARMA
2022

Pertemuan 1

1. Jelaskan sejarah framework Spring?

   Spring Framework adalah sebuah framework open source yang berbasis Java yang dibuat oleh Rod Johnson dan dirilis pada bulan Juni 2003 dengan lisensi Apache 2.0. Tujuannya adalah untuk memberikan infrastruktur yang lengkap untuk memudahkan pengembangan aplikasi Java.

   Ketika Sun Microsystems menerbitkan spesifikasi JavaBeans pada akhir tahun 1996, Rod Johnson memulai pengembangan Spring sebagai respons terhadap kesulitan yang dihadapi pengembang saat menggunakan teknologi Java Enterprise Edition (Java EE).

2. Sebutkan 5 kelebihan dan kekurangan Spring.

   Kelebihan:
   1. **Modularitas**: Spring Framework dirancang secara modular, memungkinkan pengembang untuk menggunakan hanya bagian yang diperlukan dari framework ini. Ini membuat aplikasi lebih ringan dan lebih mudah dikelola.
   2. **Inversi Kontrol (IoC)**: Dengan menggunakan prinsip IoC dan Dependency Injection, Spring memudahkan pengelolaan objek dan dependensinya, yang menghasilkan kode yang lebih bersih dan terstruktur.
   3. **Dukungan untuk Berbagai Jenis Aplikasi**: Spring dapat digunakan untuk membangun berbagai jenis aplikasi, termasuk aplikasi web, enterprise, dan aplikasi yang memerlukan keamanan serta integrasi dengan big data
   4. **Komunitas yang Besar**: Spring memiliki komunitas yang aktif dan banyak sumber daya, termasuk dokumentasi, tutorial, dan forum, yang memudahkan pengembang untuk belajar dan mendapatkan bantuan.
   5. **Integrasi yang Mudah**: Spring Framework menyediakan dukungan untuk berbagai teknologi dan framework lain, seperti Hibernate, JPA, dan Spring Boot, yang memudahkan integrasi dalam pengembangan aplikasi.

   Kekurangan:
   1. **Kurva Pembelajaran yang Curam**: Bagi pengembang baru, memahami konsep-konsep seperti IoC dan Dependency Injection bisa menjadi tantangan, sehingga memerlukan waktu untuk belajar.
   2. **Konfigurasi yang Rumit**: Meskipun Spring Boot telah menyederhanakan banyak aspek konfigurasi, beberapa pengaturan awal masih bisa menjadi rumit dan memakan waktu.
   3. **Overhead Kinerja**: Penggunaan Spring dapat menambah overhead pada aplikasi, terutama jika tidak dioptimalkan dengan baik, yang dapat mempengaruhi kinerja.
   4. **Kompleksitas**: Dengan banyaknya fitur dan modul yang tersedia, aplikasi Spring bisa menjadi kompleks, terutama jika tidak dikelola dengan baik.
   5. **Ketergantungan pada Framework**: Pengembang mungkin menjadi terlalu bergantung pada Spring, yang dapat mengurangi pemahaman mereka tentang prinsip dasar pengembangan aplikasi Java.

3. Sebutkan 5 perusahaan yang masih menggunakan framework Spring dan digunakan untuk apa?

    1. **Intuit**: Perusahaan yang dikenal dengan produk seperti TurboTax dan QuickBooks ini menggunakan Spring untuk membangun aplikasi keuangan yang kompleks dan skalabel, memanfaatkan fitur-fitur seperti Dependency Injection dan manajemen transaksi.
    2. **Zalando**: Sebagai salah satu platform e-commerce terbesar di Eropa, Zalando menggunakan Spring untuk mengembangkan aplikasi backend yang mendukung layanan mereka, termasuk pengelolaan inventaris dan sistem pemesanan.
    3. **MIT (Massachusetts Institute of Technology)**: MIT menggunakan Spring dalam berbagai proyek penelitian dan pengembangan perangkat lunak, termasuk aplikasi yang mendukung sistem pendidikan dan penelitian ilmiah.
    4. **Zillow**: Platform real estate ini memanfaatkan Spring untuk membangun aplikasi yang memungkinkan pengguna mencari dan menjual properti, serta mengelola data besar yang terkait dengan pasar real estate.
    5. **Deleokorea**: Perusahaan ini menggunakan Spring untuk mengembangkan aplikasi yang mendukung layanan pelanggan dan manajemen data, memanfaatkan kemampuan Spring dalam integrasi dengan berbagai sistem dan database.

4. Jelaskan bagian kode program yang telah kita lakukan pada act.

Pertemuan 5

Pertemuan5SpringBootApplication

```
package com.mahasiswa;

import com.mahasiswa.controller.MahasiswaController;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication
public class Pertemuan5SpringBootApplication implements CommandLineRunner{

    @Autowired
    private MahasiswaController mhsController;
    public static void main(String[] args) {
        SpringApplication.run(Pertemuan5SpringBootApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        mhsController.tampilkanMenu();
    }

}
```

- Melakukan import spring framework.
- Ketika dijalankan, metode **main** di kelas **Pertemuan5SpringBootApplication** akan dipanggil. **SpringApplication.run** digunakan untuk memulai konteks aplikasi Spring.
- Setelah konteks aplikasi berhasil dimulai, metode **run** yang diimplementasikan dari interface **CommandLineRunner** akan dieksekusi. Metode ini dirancang untuk menjalankan kode tertentu setelah aplikasi siap. Dalam hal ini, metode **run** memanggil **tampilkanMenu()** dari **MahasiswaController**, yang bertanggung jawab untuk menampilkan menu interaksi kepada pengguna.

MahasiswaController.java

```
package com.mahasiswa.controller;

import com.mahasiswa.model.ModelMahasiswa;
import com.mahasiswa.repository.MahasiswaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import java.util.List;
import java.util.Scanner;

@Controller


public class MahasiswaController {

    @Autowired
    private MahasiswaRepository mahasiswaRepository;

    public void tampilkanMenu() {
        Scanner scanner = new Scanner(System.in);
        int opsi;

        do {
            System.out.println("\nMenu:");
            System.out.println("1. Tampilkan semua mahasiswa");
            System.out.println("2. Tambah mahasiswa baru");
            System.out.println("3. Cek koneksi database");
            System.out.println("4. Keluar");
            System.out.print("Pilih opsi: ");
            opsi = scanner.nextInt();
            scanner.nextLine(); // menangkap newline

            switch (opsi) {
                case 1:
                    tampilkanSemuaMahasiswa();
                    break;
                case 2:
                    tambahMahasiswa(scanner);
                    break;
                case 3:
```

```
                cekKoneksi();
                break;
            case 4:
                System.out.println("Keluar dari program.");
                break;
            default:
                System.out.println("Opsi tidak valid, coba lagi.");
        }

    } while (opsi != 4);
}

private void tampilkanSemuaMahasiswa() {
    List<ModelMahasiswa> mahasiswaList = mahasiswaRepository.findAll();
    if (mahasiswaList.isEmpty()) {
        System.out.println("Tidak ada data mahasiswa.");
    } else {
        mahasiswaList.forEach(mahasiswa -> System.out.println(mahasiswa));
    }
}

private void tambahMahasiswa(Scanner scanner) {
    System.out.print("Masukkan NPM : ");
    String npm = scanner.nextLine();
    System.out.print("Masukkan Nama : ");
    String nama = scanner.nextLine();
    System.out.print("Masukkan Semester : ");
    int semester = scanner.nextInt();
    System.out.print("Masukkan IPK : ");
    float ipk = scanner.nextFloat();

    ModelMahasiswa mahasiswa = new ModelMahasiswa(0, npm, nama, semester, ipk);
    mahasiswaRepository.save(mahasiswa);
    System.out.println("Mahasiswa berhasil ditambahkan.");
}

private void cekKoneksi() {
    try {
        mahasiswaRepository.findAll();
        System.out.println("Koneksi ke database berhasil.");
    } catch (Exception e) {
        System.out.println("Gagal terhubung ke database.");
    }
}
}
```

- **@Controller** menunjukkan bahwa kelas ini merupakansebuah komponen Spring yang bertanggung jawab untuk menangani permintaan dan mengembalikan respon.
- **@Autowired** digunakan untuk melakukan dependency injection, yang memungkinkan Spring untuk secara otomatis menginstansiasi dan mengelola objek **MahasiswaRepository**. Ini berarti **mahasiswaRepository** akan diisi dengan implementasi yang sesuai saat aplikasi dijalankan.

- Scanner akan membaca input dari pengguna.
- Sebuah loop do-while digunakan untuk menampilkan menu dan meminta input pengguna hingga pengguna memilih untuk keluar (opsi 4).
- Menu yang ditampilkan memiliki empat opsi:
    - Tampilkan semua mahasiswa.
    - Tambah mahasiswa baru.
    - Cek koneksi database.
    - Keluar dari program.
- Input pengguna diproses menggunakan switch untuk menentukan tindakan yang sesuai berdasarkan opsi yang dipilih.

- Metode ini memanggil **mahasiswaRepository.findAll()** untuk mengambil daftar semua mahasiswa dari database.
- Jika daftar mahasiswa kosong, pesan "Tidak ada data mahasiswa." ditampilkan. Jika tidak kosong, setiap objek mahasiswa ditampilkan ke layar menggunakan **forEach**.

- **tambahMahasiswa(Scanner scanner)** digunakan untuk menambahkan mahasiswa baru ke database.
- Diinput informasi mahasiswa, seperti NPM, nama, semester, dan IPK.
- Setelah menerima input, objek **ModelMahasiswa** baru dibuat dan disimpan ke database menggunakan **mahasiswaRepository.save(mahasiswa)**.
- Pesan konfirmasi ditampilkan setelah mahasiswa berhasil ditambahkan.

- **cekKoneksi()** akan memeriksa koneksi ke database dengan memanggil **mahasiswaRepository.findAll()**.
- Jika koneksi berhasil, pesan "Koneksi ke database berhasil." ditampilkan. Jika terjadi pengecualian (exception), maka pesan "Gagal terhubung ke database." ditampilkan.

ModelMahasiswa.java

```
package com.mahasiswa.model;

import jakarta.persistence.*;


@Entity
@Table(name = "mahasiswa")

public class ModelMahasiswa {

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

    @Column(name = "npm", nullable = false, length = 8)
    private String npm;

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

    @Column(name = "semester")
    private int semester;

    @Column(name = "ipk")
    private float ipk;

    public ModelMahasiswa(){

    }

    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;
    }

    @Override
    public String toString() {
        return "Mahasiswa{" +
                "id =" + id +
                ", npm ='" + npm + '\'' +
                ", nama ='" + nama + '\'' +
                ", semester ='" + semester + '\'' +
                ", ipk ='" + ipk + '\'' +
                '}';
    }
}
```

- **@Entity**: Menandakan bahwa kelas ini adalah entitas JPA yang akan dipetakan ke tabel dalam database.
- **@Table(name = "mahasiswa")**: Menentukan nama tabel dalam database yang akan dipetakan ke entitas ini, yaitu tabel **mahasiswa**.

- Kelas **ModelMahasiswa** memiliki beberapa atribut yang merepresentasikan kolom dalam tabel **mahasiswa**:
  - **@Id**: Menandakan bahwa atribut **id** adalah primary key dari entitas ini.
  - **@GeneratedValue(strategy = GenerationType.IDENTITY)**: Menunjukkan bahwa nilai **id** akan dihasilkan secara otomatis oleh database (biasanya menggunakan auto-increment).
  - **@Column**: Digunakan untuk memetakan atribut ke kolom dalam tabel.

- Atribut-atribut dalam kelas :
  - **private int id;**: ID unik untuk setiap mahasiswa.
  - **private String npm;**: Nomor Pokok Mahasiswa.
  - **private String nama;**: Nama mahasiswa.
  - **private int semester;**: Semester yang sedang ditempuh.
  - **private float ipk;**: Indeks Prestasi Kumulatif.

- **ModelMahasiswa()**: Konstruktor default tanpa parameter.
- **ModelMahasiswa(int id, String npm, String nama, int semester, float ipk)**: Konstruktor dengan parameter untuk menginisialisasi semua atribut kelas.

- **Getter dan Setter** memungkinkan akses dan modifikasi nilai atribut dari objek ModelMahasiswa:
  - **getId(), setId(int id)**: Mengambil dan menetapkan nilai ID.
  - **getNpm(), setNpm(String npm)**: Mengambil dan menetapkan nilai NPM.
  - **getNama(), setNama(String nama)**: Mengambil dan menetapkan nilai nama.
  - **getSemester(), setSemester(int semester)**: Mengambil dan menetapkan nilai semester.
  - **getIpk(), setIpk(float ipk)**: Mengambil dan menetapkan nilai IPK

- **toString()** di-override untuk memberikan representasi string dari objek ModelMahasiswa.

MahasiswaRepository.Java

```
package com.mahasiswa.repository;

import com.mahasiswa.model.ModelMahasiswa;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MahasiswaRepository extends JpaRepository<ModelMahasiswa, Long> {

}
```

- **@Repository**: Anotasi ini menandakan bahwa interface ini adalah komponen Spring yang berfungsi sebagai repository.

- **JpaRepository<ModelMahasiswa, Long>**:
  - (**ModelMahasiswa**) menunjukkan jenis entitas yang akan dikelola oleh repository ini.
  - (**Long**) menunjukkan tipe data dari primary key entitas tersebut.

application.properties

```
# Konfigurasi MySQL Hibernate
spring.datasource.url=jdbc:mysql://localhost:3306/rpl_pert5?useSSL=false&serverTimez
one=UTC
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate settings
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

- Mengkonfigurasikan ke MySQL

Pertemuan 6

MahasiswaApp.java

```
package com.mahasiswa;

import com.mahasiswa.controller.MahasiswaController;
import com.mahasiswa.service.MahasiswaService;
import com.mahasiswa.view.MahasiswaView;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;


@SpringBootApplication
public class MahasiswaApp implements ApplicationRunner {

    @Autowired
    private MahasiswaService mahasiswaService;

    public static void main(String[] args) {
        System.setProperty("java.awt.headless", "false"); // Disable headless mode

        // Start the Spring application and get the application context
        ApplicationContext context = SpringApplication.run(MahasiswaApp.class, args);

        // Instantiate the view and inject the controller manually
        MahasiswaController controller = context.getBean(MahasiswaController.class);
        MahasiswaView mahasiswaView = new MahasiswaView(controller);
        mahasiswaView.setVisible(true);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        // Implement this method if you need to execute logic after Spring application starts
        // Otherwise, you can leave it as is.
    }
}
```

- Melakukan import Spring Framework dan komponen yang diperlukan.
- Ketika dijalankan, metode **main** di kelas **MahasiswaApp** akan dipanggil. **SpringApplication.run** digunakan untuk memulai konteks aplikasi Spring.
- Setelah aplikasi berhasil dimulai, metode **run** yang diimplementasikan dari interface **ApplicationRunner** dapat dieksekusi.
- Kelas **MahasiswaController** diambil dari konteks aplikasi dan digunakan untuk membuat instance **MahasiswaView**, yang kemudian ditampilkan kepada pengguna.

MahasiswaController.java

```
package com.mahasiswa;

import com.mahasiswa.controller.MahasiswaController;
import com.mahasiswa.service.MahasiswaService;
import com.mahasiswa.view.MahasiswaView;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;


@SpringBootApplication
public class MahasiswaApp implements ApplicationRunner {

    @Autowired
    private MahasiswaService mahasiswaService;

    public static void main(String[] args) {
        System.setProperty("java.awt.headless", "false"); // Disable headless mode

        // Start the Spring application and get the application context
        ApplicationContext context = SpringApplication.run(MahasiswaApp.class, args);

        // Instantiate the view and inject the controller manually
        MahasiswaController controller = context.getBean(MahasiswaController.class);
        MahasiswaView mahasiswaView = new MahasiswaView(controller);
        mahasiswaView.setVisible(true);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        // Implement this method if you need to execute logic after Spring application starts
        // Otherwise, you can leave it as is.
    }
}
```

- Mengimpor paket-paket yang diperlukan dari Spring Framework dan komponen aplikasi.
- Kelas **MahasiswaApp** ditandai dengan anotasi **@SpringBootApplication**, yang menandakan bahwa ini adalah aplikasi Spring Boot.
- Kelas ini mengimplementasikan interface **ApplicationRunner**, yang memungkinkan eksekusi logika setelah aplikasi dimulai.
- **@Autowired** digunakan untuk menginjeksi **MahasiswaService**, sehingga dapat digunakan dalam kelas ini.
- Metode **main** adalah titik masuk aplikasi. Di dalamnya:
    - Mengatur properti sistem untuk menonaktifkan mode headless, yang memungkinkan penggunaan antarmuka grafis.
    - Memanggil **SpringApplication.run** untuk memulai konteks aplikasi Spring.

    - Mengambil instance **MahasiswaController** dari konteks aplikasi.
    - Membuat instance **MahasiswaView** dengan menginjeksi **MahasiswaController** dan menampilkannya kepada pengguna.
- Metode **run** dapat digunakan untuk menjalankan logika tambahan setelah aplikasi siap, tetapi dalam kode ini tidak ada logika yang ditambahkan.

ModelMahasiswa.java

```
package com.mahasiswa.model;

import jakarta.persistence.*;


@Entity
@Table(name = "mahasiswa")

public class ModelMahasiswa {

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

    @Column(name = "npm", nullable = false, length = 8)
    private String npm;

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

    @Column(name = "semester")
    private int semester;

    @Column(name = "ipk")
    private float ipk;

    public ModelMahasiswa(){

    }

    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;
    }
}
```

- **@Entity** menandakan bahwa kelas ini adalah entitas JPA (Java Persistence API) yang akan dipetakan ke tabel dalam basis data.
- **@Table(name = "mahasiswa")** menunjukkan entitas ini berkaitan dengan tabel "mahasiswa" di basis data.

Atribut pada kelas:

- **@Id**: Menandakan bahwa atribut **id** adalah kunci utama dari entitas ini.
- **@GeneratedValue(strategy = GenerationType.IDENTITY)**: Mengatur agar nilai **id** dihasilkan secara otomatis oleh basis data.
- **@Column**: Digunakan untuk mendefinisikan kolom dalam tabel basis data, termasuk nama kolom, apakah kolom tersebut boleh bernilai null, dan panjang maksimum untuk kolom string.
    - **npm**: Nomor pokok mahasiswa, tidak boleh null dan memiliki panjang maksimum 8 karakter.

- **nama**: Nama mahasiswa, tidak boleh null dan memiliki panjang maksimum 50 karakter.
- **semester**: Semester mahasiswa, bertipe integer.
- **ipk**: Indeks prestasi kumulatif mahasiswa, bertipe float.

- Terdapat metode getter dan setter untuk setiap atribut, yang memungkinkan akses dan modifikasi nilai atribut dari luar kelas.

ModelTabelMahasiswa.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.mahasiswa.model;

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



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
    }

    @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
    }
}
```

- **mahasiswaList**: Daftar yang menyimpan objek-objek **ModelMahasiswa**, yang berisi data mahasiswa.
- **columnNames**: Sebuah array string yang berisi nama-nama kolom yang akan ditampilkan di tabel, yaitu "ID", "NPM", "Nama", "Semester", dan "IPK".

- **ModelTabelMahasiswa(List<ModelMahasiswa> mahasiswaList)**: Konstruktor ini menerima daftar mahasiswa dan menginisialisasi atribut **mahasiswaList**.

- **getRowCount()**: Mengembalikan jumlah baris dalam tabel, yang sama dengan ukuran daftar **mahasiswaList**.
- **getColumnCount()**: Mengembalikan jumlah kolom dalam tabel, yang sama dengan panjang array **columnNames**.
- **getValueAt(int rowIndex, int columnIndex)**: Mengembalikan nilai dari sel tertentu berdasarkan indeks baris dan kolom.
    - Mengambil objek **ModelMahasiswa** dari **mahasiswaList** berdasarkan **rowIndex**.
    - Menggunakan **switch** untuk menentukan nilai yang akan dikembalikan berdasarkan **columnIndex**.
- **getColumnName(int column)**: Mengembalikan nama kolom berdasarkan indeks kolom yang diberikan.
- **isCellEditable(int rowIndex, int columnIndex)**: Mengembalikan **false**, yang berarti semua sel dalam tabel tidak dapat diedit oleh pengguna.

MahasiswaRepository.Java

```
package com.mahasiswa.repository;

import com.mahasiswa.model.ModelMahasiswa;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MahasiswaRepository extends JpaRepository<ModelMahasiswa, Integer> {
    public Object findById(int id);

    public void deleteById(int id);

}
```

- **@Repository**: Menandakan bahwa interface ini adalah komponen Spring yang berfungsi sebagai repository.

MahasiswaService.java

```
package com.mahasiswa.service;

import com.mahasiswa.model.ModelMahasiswa;
import com.mahasiswa.repository.MahasiswaRepository;
import jakarta.transaction.Transactional;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MahasiswaService {

    @Autowired
    private MahasiswaRepository repository;

    public void addMhs(ModelMahasiswa mhs) {
        repository.save(mhs);
    }

    public ModelMahasiswa getMhs(int id) {
        ModelMahasiswa mahasiswa = (ModelMahasiswa) repository.findById(id);
        return mahasiswa != null ? mahasiswa : null;
    }

    public void updateMhs(ModelMahasiswa mhs) {
        repository.save(mhs);
    }

    @Transactional
    public void deleteMhs(int id) {
```

```
        repository.deleteById(id);
    }

    public List<ModelMahasiswa> getAllMahasiswa() {
        return repository.findAll();
    }
}
```

- **@Service**: Menandakan bahwa kelas ini adalah komponen layanan dalam konteks Spring, sehingga dapat diidentifikasi dan dikelola oleh Spring Container.

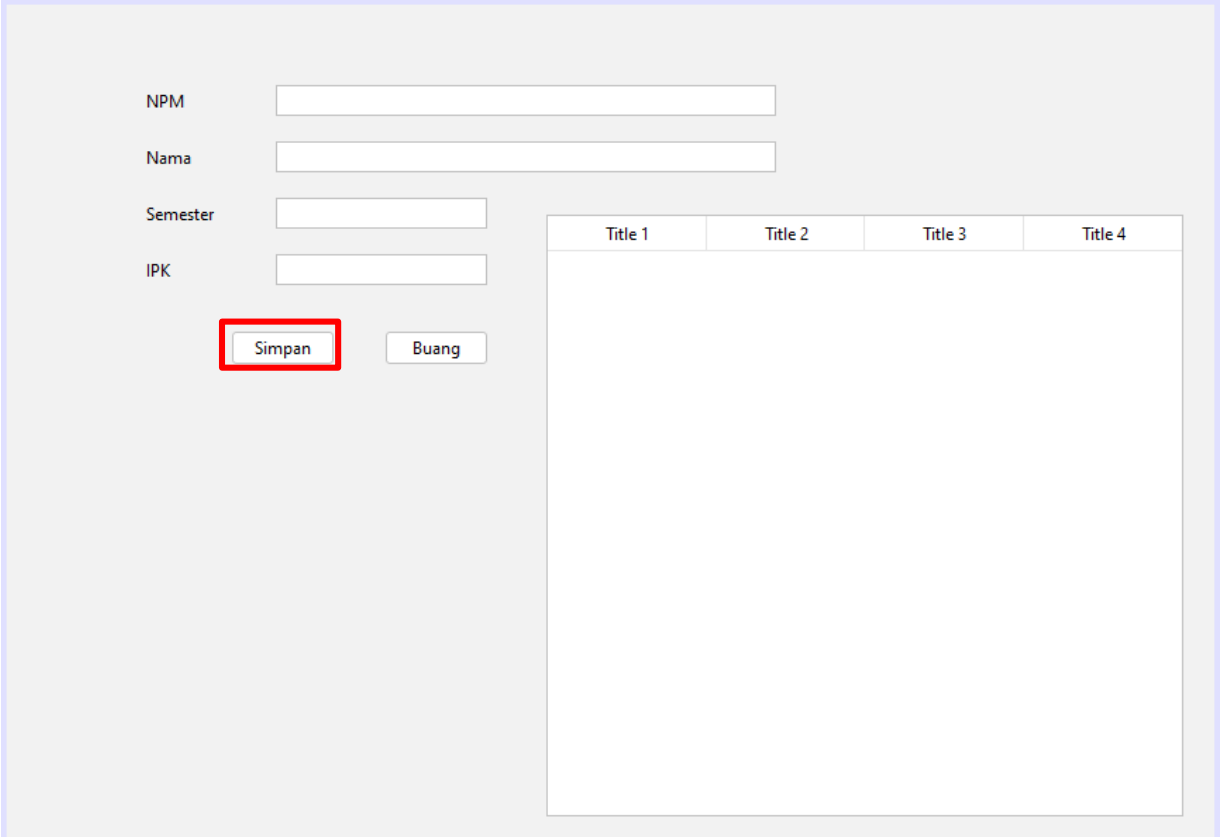
- **@Autowired**: Menginjeksi **MahasiswaRepository**, yang merupakan antarmuka untuk melakukan operasi Create, Read, Update, Delete pada entitas **ModelMahasiswa**.

Metode dalam Kelas:

- **addMhs(ModelMahasiswa mhs)**:
    - Metode ini menerima objek **ModelMahasiswa** sebagai parameter dan menyimpannya ke dalam basis data dengan memanggil metode **save** dari **repository**.
- **getMhs(int id)**:
    - Metode ini mengambil mahasiswa berdasarkan **id** yang diberikan.
    - Menggunakan **repository.findById(id)** untuk mencari mahasiswa. Jika mahasiswa ditemukan, objek tersebut dikembalikan; jika tidak, mengembalikan **null**.
- **updateMhs(ModelMahasiswa mhs)**:
    - Memanggil **repository.save(mhs)**, yang akan menyimpan objek **mhs** ke dalam basis data. Jika **mhs** sudah ada, maka data tersebut akan diperbarui.
- **deleteMhs(int id)**:
    - Metode ini digunakan untuk menghapus mahasiswa berdasarkan **id**.
    - **@Transactional** menandakan bahwa metode ini harus dijalankan dalam konteks transaksi. Jika terjadi kesalahan, perubahan dapat dibatalkan.
    - Memanggil **repository.deleteById(id)** untuk menghapus mahasiswa dari basis data.
- **getAllMahasiswa()**:
    - Memanggil **repository.findAll()** untuk mendapatkan semua entitas **ModelMahasiswa**.

MahasiswaView.java

simpanButton

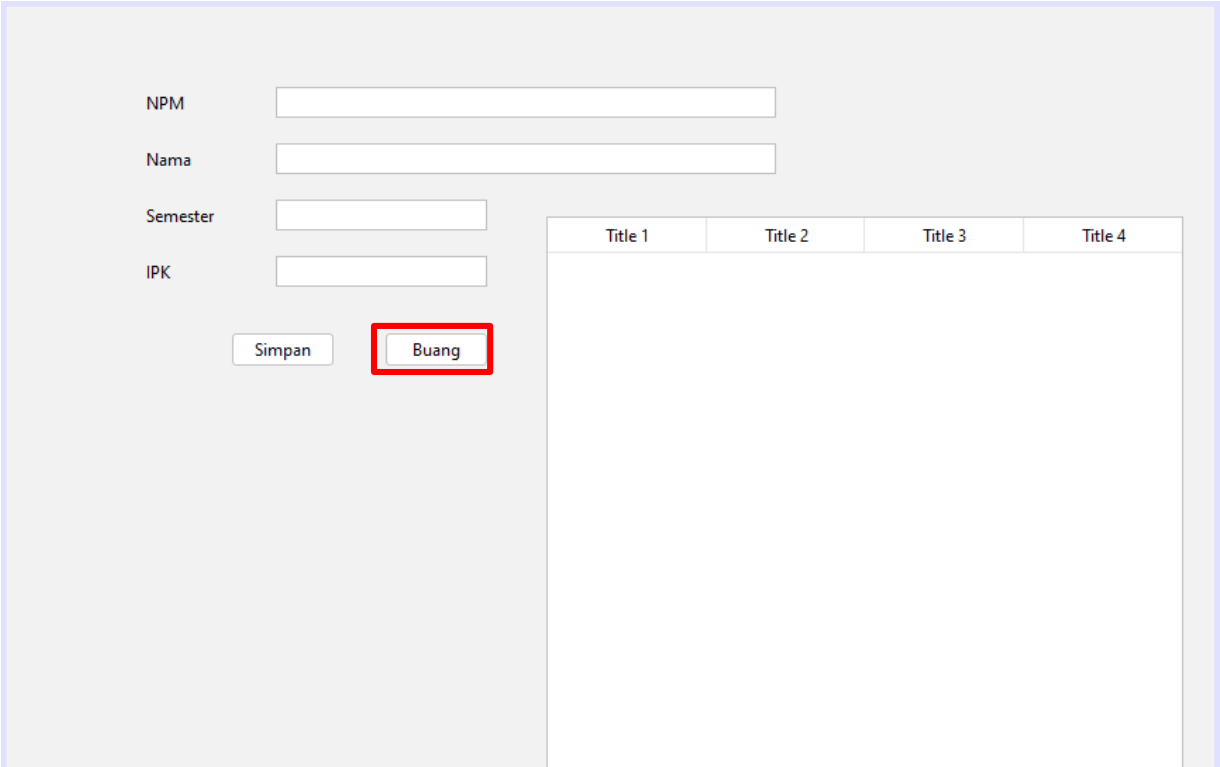



```
private void simpanButtonActionPerformed(java.awt.event.ActionEvent evt) {
  String npm = getNpmField().getText();
  String nama = getNamaField().getText();
  int semester = Integer.parseInt(getSemesterField().getText());
  float ipk = Float.parseFloat(getIpkField().getText());
  ModelMahasiswa mahasiswa = new ModelMahasiswa(0, npm, nama, semester, ipk);
  System.out.println(mahasiswa.getIpk());
  System.out.println(mahasiswa.getNama());
  System.out.println(mahasiswa.getSemester());
  System.out.println(mahasiswa.getNpm());

  controller.addMahasiswa(mahasiswa);
  loadMahasiswaTable();
 }
```

- Mengambil nilai dari field input untuk NPM, nama, semester, dan IPK menggunakan metode **getText()**.
- Mengonversi nilai semester dari string ke integer menggunakan **Integer.parseInt()**.
- Mengonversi nilai IPK dari string ke float menggunakan **Float.parseFloat()**.
- Membuat instance baru dari **ModelMahasiswa**.
- Mencetak nilai IPK, nama, semester, dan NPM ke konsol untuk tujuan debugging atau verifikasi.
- Memanggil metode **addMahasiswa(mahasiswa)** dari **controller** untuk menyimpan objek **mahasiswa** ke dalam basis data atau sistem.
- Memanggil **loadMahasiswaTable()** untuk memperbarui tampilan tabel mahasiswa.

buangButton




```
    private void buangButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JTextField idField = new JTextField(5);

    // Membuat panel untuk menampung JTextField
    JPanel panel = new JPanel();
    panel.add(new JLabel("Masukkan ID yang ingin dihapus:"));
    panel.add(idField);
    // Menampilkan dialog box dengan JTextField, tombol OK, dan Cancel
    int result = JOptionPane.showConfirmDialog(null, panel,
        "Hapus Mahasiswa", JOptionPane.OK_CANCEL_OPTION,
JOptionPane.PLAIN_MESSAGE);
    // Jika tombol OK ditekan
    if (result == JOptionPane.OK_OPTION) {
        try {
            // Mengambil input ID dan memanggil metode deleteMhs
            int id = Integer.parseInt(idField.getText());
            controller.deleteMahasiswa(id);
            JOptionPane.showMessageDialog(null, "Data berhasil dihapus.", "Sukses",
JOptionPane.INFORMATION_MESSAGE);
        } catch (NumberFormatException e) {
            // Menangani error jika ID yang dimasukkan bukan angka
            JOptionPane.showMessageDialog(null, "ID harus berupa angka.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
    loadMahasiswaTable();
    }
```

- Membuat **JTextField** untuk menerima input ID mahasiswa yang ingin dihapus.
- Membuat **JPanel** untuk menampung label dan **JTextField**, yang akan ditampilkan dalam dialog.
- Menggunakan **JOptionPane.showConfirmDialog** untuk menampilkan dialog yang berisi panel dengan input field. Dialog ini memiliki tombol OK dan Cancel.
- Jika pengguna menekan tombol OK (**result == JOptionPane.OK_OPTION**), maka program akan melanjutkan untuk mengambil input ID.
- Mengambil teks dari **idField**, mengonversinya menjadi integer, dan memanggil metode **deleteMahasiswa(id)** dari **controller** untuk menghapus mahasiswa dengan ID tersebut.
- Jika penghapusan berhasil, menampilkan pesan sukses menggunakan **JOptionPane.showMessageDialog**.
- Jika terjadi **NumberFormatException** (misalnya, jika pengguna memasukkan nilai yang bukan angka), menampilkan pesan kesalahan yang sesuai.
- Memanggil **loadMahasiswaTable()** untuk memperbarui tampilan tabel mahasiswa di antarmuka pengguna setelah penghapusan.