LAPORAN AKHIR PRAKTIKUM

Mata Praktikum	: RPL 2
Kelas	: 4IA06
Praktikum ke-	: 3
Tanggal	: 29 October 2024
Materi	: Konsep MVC
NPM	: 50421535
Nama	: Fizri Rosdiansyah
Ketua Asisten	: Gilbert Jefferson Faozato Mendrofa
Paraf Asisten	:
Nama Asisten	:
Jumlah Lembar	: 10 Lembar

LABORATORIUM TEKNIK INFORMATIKA

UNIVERSITAS GUNADARMA

2024

Rekayasa Perangkat Lunak 2

Soal

1. Jelaskan satu per satu Codingan kalian dari hasil screenshot Activity

Jawab

1.

A.Mahasiswa Controller

```
11      public void displayMahasiswaList(List<ModelMahasiswa> mahasiswaList){
12          if(mahasiswaList.isEmpty()){
13              System.out.println("Tidak ada data mahasiswa");
14          }else{
15              System.out.println("");
16              System.out.println("============================");
17              for(ModelMahasiswa m: mahasiswaList){
18                  System.out.println("ID          : " + m.getId());
19                  System.out.println("NPM         : " + m.getNpm());
20                  System.out.println("NAMA        : " + m.getNama());
21                  System.out.println("SEMESTER    : " + m.getSemester());
22                  System.out.println("IPK         : " + m.getIpk());
23                  System.out.println("============================");
24              }
25              displayMessage("Mahasiswa berhasil ditampilkan");
26          }
27      }
```

Metode ini menampilkan daftar mahasiswa:

- Jika mahasiswaList kosong, ditampilkan pesan "Tidak ada data mahasiswa".
- Jika ada data, iterasi dilakukan untuk menampilkan detail tiap mahasiswa (ID, NPM, Nama, Semester, IPK).
- Menampilkan pesan "Mahasiswa berhasil ditampilkan" setelah iterasi selesai.

```
    public void displayMessage(String message){
        System.out.println(message);
    }
```

Metode ini hanya menampilkan pesan yang diteruskan sebagai parameter ke konsol.

```
33      public MahasiswaController(MahasiswaDAO mahasiswaDAO){
34          this.mahasiswaDAO = mahasiswaDAO;
35      }
```

Konstruktor ini digunakan untuk menginisialisasi MahasiswaController dengan objek MahasiswaDAO, yang memungkinkan interaksi dengan database.

```
public void checkDatabaseConnection(){
    boolean isConnected = mahasiswaDAO.cekKoneksi();
    if (isConnected){
        displayMessage("Koneksi ke db berhasil");
    } else{
        displayMessage("Koneksi DB gagal");
    }
}
```

Metode ini mengecek koneksi ke database:

- Memanggil metode cekKoneksi() dari mahasiswaDAO untuk mendapatkan status koneksi.
- Menampilkan pesan "Koneksi ke db berhasil" jika terkoneksi, atau "Koneksi DB gagal" jika tidak.

```
public void displayAllMahasiswa(){
    List<ModelMahasiswa> mahasiswaList = mahasiswaDAO.getAllMahasiswa();
    displayMahasiswaList(mahasiswaList);
}
```

Metode ini menampilkan seluruh data mahasiswa:

- Mengambil semua data mahasiswa dengan getAllMahasiswa dari mahasiswaDAO.
- Menampilkan data tersebut menggunakan displayMahasiswaList.

```
public void addMahasiswa(String npm, String nama, int semester, float ipk){
    ModelMahasiswa mahasiswaBaru = new ModelMahasiswa(0, npm, nama, semester, ipk);
    mahasiswaDAO.addMahasiswa(mahasiswaBaru);
    displayMessage("Mahasiswa berhasil ditambahkan");
}
```

Metode ini menambahkan mahasiswa baru ke database:

- Membuat objek ModelMahasiswa baru dengan parameter NPM, Nama, Semester, dan IPK.
- Menyimpan data baru ini ke database menggunakan addMahasiswa dari mahasiswaDAO.
- Menampilkan pesan "Mahasiswa berhasil ditambahkan" di konsol.

```
public void updateMahasiswa(int id, String npm, String nama, int semester, float ipk){
    ModelMahasiswa mahasiswaBaru = new ModelMahasiswa(id, npm, nama, semester, ipk);
    mahasiswaDAO.updateMahasiswa(mahasiswaBaru);
    displayMessage("Mahasiswa berhasil diperbaharui");
}
```

Metode ini memperbarui data mahasiswa yang ada berdasarkan ID:

- Membuat objek ModelMahasiswa dengan ID dan data baru.
- Memperbarui data mahasiswa di database menggunakan updateMahasiswa dari mahasiswaDAO.
- Menampilkan pesan "Mahasiswa berhasil diperbaharui" setelah pembaruan berhasil.

```
public void deleteMahasiswa(int id){
    mahasiswaDAO.deleteMahasiswa(id);
    displayMessage("Mahasiswa Berhasil Dihapus!");
}
```

Metode ini menghapus data mahasiswa berdasarkan ID:

- Memanggil deleteMahasiswa dari mahasiswaDAO untuk menghapus data dari database.
- Menampilkan pesan "Mahasiswa Berhasil Dihapus!" setelah penghapusan.

```
    public void closeConnection(){
        mahasiswaDAO.closeConnection();
    }
}
```

Metode ini menutup koneksi database dengan memanggil closeConnection pada mahasiswaDAO.

B.MahasiswaDAO

```
7   public class MahasiswaDAO {
8       private Connection connection;
9
10      public MahasiswaDAO(){
11          try{
12              Class.forName("com.mysql.cj.jdbc.Driver");
13              connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mvc_db", "root", "");
            }catch(Exception e){
                e.printStackTrace();
16          }
17      }
18
```

Konstruktor ini digunakan untuk menginisialisasi koneksi database:

- Class.forName("com.mysql.cj.jdbc.Driver") memuat driver MySQL JDBC.
- DriverManager.getConnection(...) digunakan untuk menghubungkan ke database mvc_db di localhost dengan username root dan tanpa password
- Jika terjadi kesalahan, e.printStackTrace() mencetak detail error ke konsol.

```
19      public boolean cekKoneksi(){
20          try{
21              if(connection != null && !connection.isClosed()){
22                  return true;
23              }
24          }catch (SQLException e){
                e.printStackTrace();
26          }
27          return false;
28      }
```

Metode ini mengecek apakah koneksi ke database berhasil:

- Mengembalikan true jika koneksi aktif dan belum ditutup.
- Jika ada kesalahan SQL, detailnya akan dicetak, dan metode mengembalikan false.

```
30      public void addMahasiswa(ModelMahasiswa mahasiswa){
31          String sql = "INSERT INTO mahasiswa (npm, nama, semester, ipk) VALUES (?, ?, ?, ?)";
32          try{
33              PreparedStatement pstmt = connection.prepareStatement(sql);
34              pstmt.setString(1, mahasiswa.getNpm());
35              pstmt.setString(2, mahasiswa.getNama());
36              pstmt.setInt(3, mahasiswa.getSemester());
37              pstmt.setFloat(4, mahasiswa.getIpk());
38              pstmt.executeUpdate();
39          } catch (SQLException e){
                e.printStackTrace();
41          }
42      }
```

Metode ini menambahkan data mahasiswa baru ke tabel mahasiswa:

- Menggunakan PreparedStatement untuk mencegah SQL injection.
- Mengisi nilai npm, nama, semester, dan ipk dari objek ModelMahasiswa yang diterima sebagai parameter.
- executeUpdate() mengeksekusi query untuk memasukkan data ke database.

```
    public List<ModelMahasiswa> getAllMahasiswa(){
        List<ModelMahasiswa> mahasiswaList = new ArrayList<>();
        String sql = "SELECT * FROM mahasiswa";
        try{
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(sql);
           while(rs.next()){
               mahasiswaList.add(new ModelMahasiswa(
                       rs.getInt("id"),
                       rs.getString("npm"),
                       rs.getString("nama"),
                       rs.getInt("semester"),
                       rs.getFloat("ipk")
               ));
           }
        } catch(SQLException e){
            e.printStackTrace();
        }
        return mahasiswaList;
    }
```

Metode ini mengambil semua data dari tabel mahasiswa dan mengembalikannya dalam bentuk List<ModelMahasiswa>:

- Menggunakan Statement dan ResultSet untuk mengeksekusi dan menyimpan hasil query SELECT * FROM mahasiswa.
- Menggunakan while(rs.next()) untuk mengiterasi hasil query, membuat objek ModelMahasiswa untuk setiap baris data, dan menambahkannya ke mahasiswaList.
- Mengembalikan mahasiswaList yang berisi data semua mahasiswa.

```
65  public void updateMahasiswa(ModelMahasiswa mahasiswa){
66      String sql = "UPDATE mahasiswa SET npm = ?, nama = ?, semester = ?, ipk = ? WHERE id = ?";
67      try{
68          PreparedStatement pstmt = connection.prepareStatement(sql);
69          pstmt.setString(1, mahasiswa.getNpm());
70          pstmt.setString(2, mahasiswa.getNama());
71          pstmt.setInt(3, mahasiswa.getSemester());
72          pstmt.setFloat(4, mahasiswa.getIpk());
73          pstmt.setInt(5, mahasiswa.getId());
74      } catch(SQLException e){
            e.printStackTrace();
76      }
77  }
```

Metode ini memperbarui data mahasiswa berdasarkan ID yang diberikan:

- Menggunakan PreparedStatement dengan query UPDATE untuk memperbarui data mahasiswa.
- Mengisi nilai npm, nama, semester, ipk, dan id dari objek ModelMahasiswa yang diberikan sebagai parameter.
- executeUpdate() mengeksekusi query untuk memperbarui data di database.

```
79      public void deleteMahasiswa(int id){
80          String sql = "DELETE FROM mahasiswa WHERE id = ?";
81          try{
82              PreparedStatement pstmt = connection.prepareStatement(sql);
83              pstmt.setInt(1, id);
84              pstmt.executeUpdate();
85          } catch(SQLException e){
                e.printStackTrace();
87          }
88      }
```

Metode ini menghapus data mahasiswa berdasarkan ID:

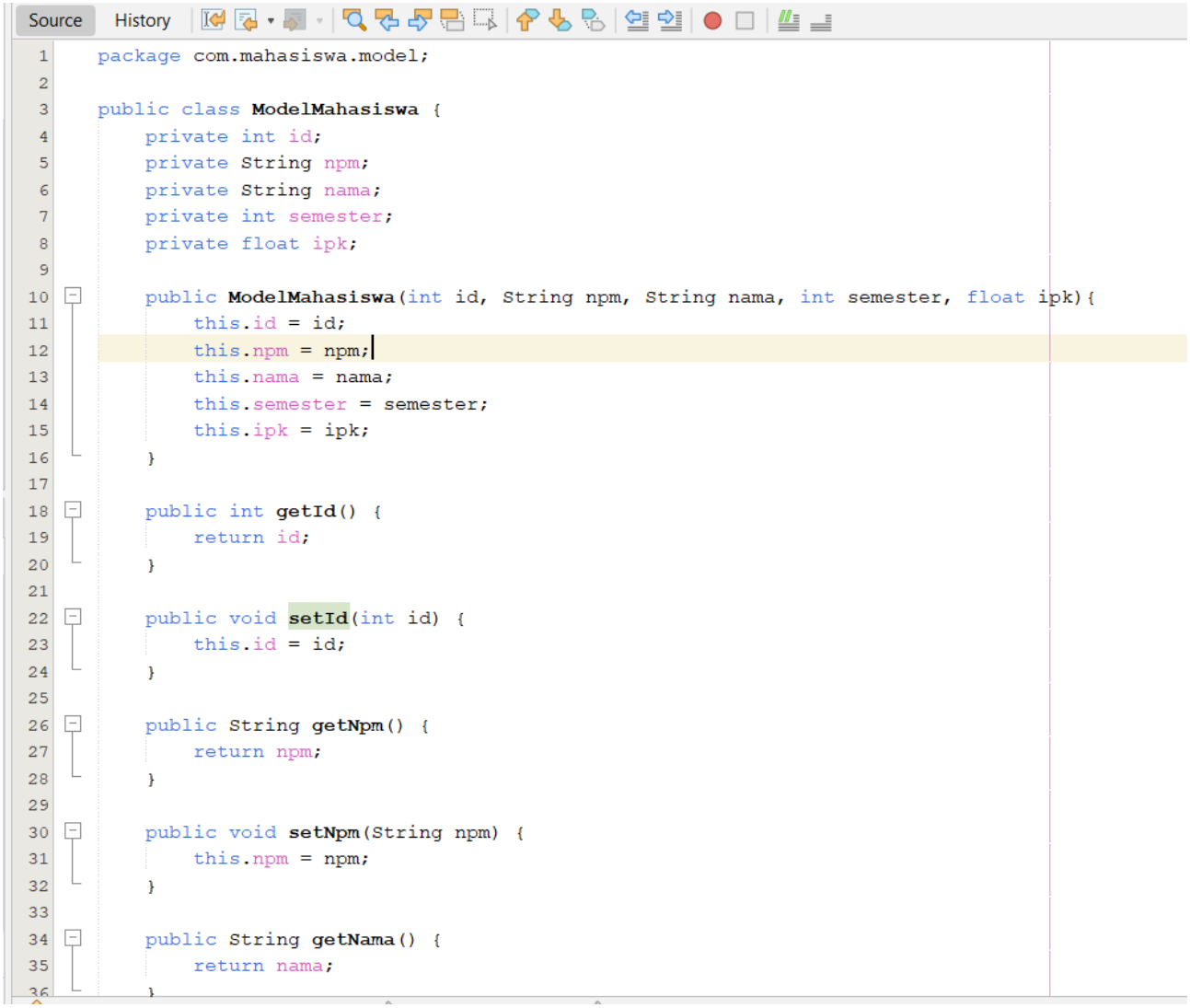
- Menggunakan PreparedStatement untuk mencegah SQL injection.
- Memasukkan nilai id sebagai parameter dalam query DELETE FROM mahasiswa WHERE id = ?.
- executeUpdate() mengeksekusi query untuk menghapus data dari database.

```
90      public void closeConnection(){
91          try{
92              if(connection != null){
93                  connection.close();
94              }
95          }catch(SQLException e){
                e.printStackTrace();
97          }
98      }
99  }
```

Metode ini menutup koneksi database jika masih terbuka:

- Mengecek apakah connection tidak null, lalu memanggil close() untuk menutup koneksi.
- Jika terjadi kesalahan saat menutup koneksi, detailnya dicetak ke konsol.

C.ModelMahasiswa



```java
package com.mahasiswa.model;

public class ModelMahasiswa {
    private int id;
    private String npm;
    private String nama;
    private int semester;
    private float ipk;

    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;
    }
```

Kelas ModelMahasiswa adalah kelas model yang merepresentasikan entitas mahasiswa dalam sistem. Kelas ini memiliki beberapa atribut (atau field) yang mendeskripsikan data mahasiswa:

- id (int): Menyimpan ID unik untuk setiap mahasiswa.
- npm (String): Menyimpan Nomor Pokok Mahasiswa (NPM), yang umumnya adalah identifikasi unik mahasiswa di universitas.
- nama (String): Menyimpan nama mahasiswa.
- semester (int): Menyimpan semester aktif mahasiswa.
- ipk (float): Menyimpan IPK (Indeks Prestasi Kumulatif) mahasiswa.

Field ini diatur sebagai private, sehingga hanya dapat diakses dan diubah melalui metode getter dan setter yang disediakan.

Konstruktor disini memungkinkan untuk membuat objek ModelMahasiswa baru dengan semua atribut yang diperlukan. Konstruktor ini menerima lima parameter:

- id, npm, nama, semester, dan ipk.
- Masing-masing parameter ini kemudian diinisialisasi ke field yang sesuai di dalam kelas menggunakan this.

getter dan setter digunakan untuk mengakses dan memodifikasi nilai-nilai dari field id, npm, nama, semester, dan ipk dari luar kelas ini.

D.MahasiswaView

```
public static void main(String[] args) {
    MahasiswaDAO mahasiswaDAO = new MahasiswaDAO();
    MahasiswaController mahasiswaController = new MahasiswaController(mahasiswaDAO);
```

- Membuat objek MahasiswaDAO untuk mengelola interaksi dengan database.
- Membuat objek MahasiswaController, yang akan menerima mahasiswaDAO sebagai parameter agar dapat menggunakan akses database.

```
Scanner scanner = new Scanner(System.in);
int pilihan;

while(true){
    System.out.println("Menu :");
    System.out.println("1. Tampilkan Semua Mahasiswa");
    System.out.println("2. Tambah Mahasiswa");
    System.out.println("3. Update Mahasiswa");
    System.out.println("4. Hapus Mahasiswa");
    System.out.println("5. Cek Koneksi Database");
    System.out.println("6. Keluar");
    System.out.println("PILIH OPSI: ");
    pilihan = scanner.nextInt();
    scanner.nextLine();
```

- **Scanner**: Digunakan untuk menerima input pengguna dari konsol.
- **Looping Menu**: Menampilkan menu berulang kali dengan pilihan fitur aplikasi.
- **Pilihan Input**: Meminta input nomor dari pengguna untuk memilih opsi pada menu.

```
switch(pilihan){
    case 1:
        mahasiswaController.displayAllMahasiswa();
        break;
    case 2:
        System.out.println("Masukan NPM: ");
        String npm = scanner.next();
        System.out.println("Masukan Nama: ");
        String nama = scanner.next();
        System.out.println("Masukan Semester: ");
        int semester = scanner.nextInt();
        System.out.println("Masukan IPK: ");
        float ipk = scanner.nextFloat();

        mahasiswaController.addMahasiswa(npm, nama, semester, ipk);
        break;
    case 3:
        System.out.println("Masukan ID Mahasiswa: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        System.out.println("Masukan NPM: ");
        String npmBaru = scanner.next();
        System.out.println("Masukan Nama: ");
        String namaBaru = scanner.next();
        System.out.println("Masukan Semester");
        int semesterBaru = scanner.nextInt();
        System.out.println("Masukan IPK: ");
        float ipkBaru = scanner.nextFloat();

        mahasiswaController.updateMahasiswa(id, npmBaru, namaBaru, semesterBaru, ipkBaru);
        break;
```

- **case 1**: Menampilkan semua data mahasiswa dengan memanggil displayAllMahasiswa() pada mahasiswaController.
- **case 2**: Menambahkan data mahasiswa baru. Program meminta npm, nama, semester, dan ipk dari pengguna, kemudian memanggil addMahasiswa() untuk menyimpan data baru.
- **case 3**: Memperbarui data mahasiswa berdasarkan id. Program meminta data baru dari pengguna (npm, nama, semester, ipk) dan memanggil updateMahasiswa() pada mahasiswaController.

```
                case 4:
                    System.out.println("Masukan ID Mahasiswa yg Ingin Dihapus: ");
                    int idHapus = scanner.nextInt();
                    mahasiswaController.deleteMahasiswa(idHapus);
                    break;
                case 5:
                    mahasiswaController.checkDatabaseConnection();
                    break;
                case 6:
                    mahasiswaController.closeConnection();
                    System.out.println("Program selesai");
                    return;
                default:
                    System.out.println("Input Tidak Valid");
            }
        }
    }
}
```

- **case 4**: Menghapus data mahasiswa berdasarkan id. Program meminta id dari pengguna dan memanggil deleteMahasiswa() pada mahasiswaController.
- **case 5**: Memeriksa koneksi database dengan memanggil checkDatabaseConnection() pada mahasiswaController.
- **case 6**: Mengakhiri program dengan menutup koneksi ke database (closeConnection()) dan menampilkan pesan "Program selesai".
- **default**: Menampilkan pesan "Input Tidak Valid" jika pengguna memasukkan angka yang tidak ada dalam menu.

E.pom.xml

```
    <dependencies>
        <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
        </dependency>

    </dependencies>
</project>
```

Kegunaan dari kode ini adalah untuk mengimpor MySQL Connector/J ke dalam proyek Java. Dengan adanya dependensi ini, netbeans dapat menggunakan API JDBC untuk menghubungkan aplikasi Java dengan database MySQL, melakukan operasi seperti query, pembaruan, dan pengelolaan data.