# Sinau Java Desktop 2 - Advanced

## [PERTEMUAN 1]

Selamat datang disesi Java Advanced. Pada kesempatan kali ini kita akan mendalami tentang data access design pattern dan reporting di Java.

## DAO / Data Accessing Object

DAO merupakan sebuah design pattern yang berguna untuk mengakses data didalam database. Pada modul Java basic 1 kita telah belajar mengakses data menggunakan JDBC dan prepare statement, dan pada kesempatan kali ini kita akan menggunakan pendekatan design pattern DAO dalam mengakses data.

Dao pattern berisi semua kode untuk mengakses data, seperti query. Lapisan lebih atas tidak boleh tahu bagaimana akses data diterapkan, lapisan lainya hanya perlu tahu fungsionalitas dari suatu method di dalam DAO class, tidak perlu tahu bagimana method tersebut diimplementasikan. Class DAO akan mempunyai method seperti save, delete, getById atau getAll. Semua table harus dipetakan kedalam class, jadi satu buah table akan mempunyai satu buah class DAO.

Dengan konsep ini kita dapat melihat skema database tanpa perlu masuk dan melihat kedalam database, cukup melihat class-class DAO yang telah dibuat.

### Database

Untuk praktek kita kali ini kita buat terlebih dahulu database dan table yang akan kita panggil di Java.

```
CREATE DATABASE sinau;

USE sinau;

CREATE TABLE `sinau`.`mahasiswa` (
  `nim` VARCHAR(15) NOT NULL,
  `nama` VARCHAR(50) NOT NULL,
  `nohp` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`nim`));
```

### Model

Dari table yang telah kita buat kita petakan ke dalam object/model di Java. Kita letakkan class ini kedalam package `model`.

```
public class Mahasiswa {

    private String nim;
    private String nama;
    private String nohp;

    public String getNim() {
        return nim;
    }
    public void setNim(String nim) {
        this.nim = nim;
    }
    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public String getNohp() {
        return nohp;
    }
    public void setNohp(String nohp) {
        this.nohp = nohp;
    }

}
```

### Class DAO

Kita buat sebuah interface untuk mendefinisikan fungsi-fungsi apasaja yang dimiliki oleh class DAO nantinya. Kita letakkan ke dalam package `dao`.

```
import java.sql.SQLException;
import java.util.List;

import javaadvanced.session1.model.Mahasiswa;

public interface MahasiswaDAO {

    public Mahasiswa saveOrUpdate(Mahasiswa mahasiswa) throws SQLException;
    public Mahasiswa delete(Mahasiswa mahasiswa) throws SQLException;
    public Mahasiswa getById(String nim) throws SQLException;
    public List<Mahasiswa> getAll() throws SQLException;

}
```

Dari interface DAO tersebut kita implementasikan ke dalam class DAO.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javaadvanced.session1.model.Mahasiswa;

public class MahasiswaDAOImpl implements MahasiswaDAO {

    private Connection connection;
    private PreparedStatement insertStatement;
    private PreparedStatement updateStatement;
    private PreparedStatement deleteStatement;
    private PreparedStatement getByIdStatement;
    private PreparedStatement getAllStatement;

    private final String INSERT_QUERY = "insert into mahasiswa(nim,nama,nohp) values(?,?,?)'
    private final String UPDATE_QUERY = "update mahasiswa set nama=?, nohp=? where nim=?";
    private final String DELETE_QUERY = "delete from mahasiswa where nim=?";
    private final String GET_BY_ID_QUERY = "select nim,nama,nohp from mahasiswa where nim=?'
    private final String GET_ALL_QUERY = "select nim,nama,nohp from mahasiswa";

    public void setConnection(Connection connection) throws SQLException{
        this.connection = connection;
        insertStatement = this.connection.prepareStatement(INSERT_QUERY);
        updateStatement = this.connection.prepareStatement(UPDATE_QUERY);
        deleteStatement = this.connection.prepareStatement(DELETE_QUERY);
        getByIdStatement = this.connection.prepareStatement(GET_BY_ID_QUERY);
        getAllStatement = this.connection.prepareStatement(GET_ALL_QUERY);
    }

    @Override
    public Mahasiswa saveOrUpdate(Mahasiswa mahasiswa, boolean isUpdate) throws SQLExceptior
        if (!isUpdate) {
            insertStatement.setString(1, mahasiswa.getNim());
            insertStatement.setString(2, mahasiswa.getNama());
            insertStatement.setString(3, mahasiswa.getNohp());
            insertStatement.executeUpdate();
        } else {
            updateStatement.setString(1, mahasiswa.getNama());
            updateStatement.setString(2, mahasiswa.getNohp());
            updateStatement.setString(3, mahasiswa.getNim());
```

```
            updateStatement.executeUpdate();
        }

        return mahasiswa;
    }

    @Override
    public Mahasiswa delete(Mahasiswa mahasiswa) throws SQLException {
        deleteStatement.setString(1, mahasiswa.getNim());
        deleteStatement.executeUpdate();

        return mahasiswa;
    }

    @Override
    public Mahasiswa getById(String nim) throws SQLException {
        getByIdStatement.setString(1, nim);
        ResultSet rs = getByIdStatement.executeQuery();

        if (rs.next()) {
            Mahasiswa mahasiswa = new Mahasiswa();
            mahasiswa.setNim(rs.getString("nim"));
            mahasiswa.setNama(rs.getString("nama"));
            mahasiswa.setNohp(rs.getString("nohp"));

            return mahasiswa;
        }

        return null;
    }

    @Override
    public List<Mahasiswa> getAll() throws SQLException {
        List<Mahasiswa> mahasiswa = new ArrayList<Mahasiswa>();
        ResultSet rs = getAllStatement.executeQuery();
        while(rs.next()){
            Mahasiswa m = new Mahasiswa();
            m.setNim(rs.getString("nim"));
            m.setNama(rs.getString("nama"));
            m.setNohp(rs.getString("nohp"));

            mahasiswa.add(m);
        }

        return mahasiswa;
    }
```

```
}
```

### Service

Dari class DAO yang telah kita buat kita petakan lagi ke dalam service. Service ini sangat berguna untuk membuat aplikasi kita menjadi transaction ready / mendukung proses transaction. Transaction adalah 1 set perintah yang akan dijalankan ke dalam database dan ketika ada 1 instruksi yang gagal maka semua perintah akan dirollback.

Misal ketika seorang nasabah mengirim uang ke nasabah lain, maka aplikasi akan mengurangi uang nasabah pengirim, kemudian mencatat pengiriman uang ke history, menambahkan uang ke nasabah penerima dan mencatat penerimaan uang ke history. Ada 4 buah operasi disini. Jika tidak menggunakan transaction, ketika ada kegagalan pada pengiriman uang makan proses yang telah berjalan tidak akan dirollback (nasabah mengirim uang, saldo tetap terpotong tetapi uang tidak diterima oleh nasabah yang menerima). Namun bila menggunakan transaction apabila ada kegagalan maka proses yang sebelumnya akan dirollback sehingga uang nasabah pengirim akan dipotong jika uang berhasil dikirimkan ke nasabah yang menerima.

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.sql.DataSource;

import javaadvanced.session1.dao.MahasiswaDAO;
import javaadvanced.session1.dao.MahasiswaDAOImpl;
import javaadvanced.session1.model.Mahasiswa;

public class MahasiswaService {

    private MahasiswaDAO mahasiswaDAO;
    private Connection connection;

    public void setDataSource(DataSource dataSource){
        try {
            connection = dataSource.getConnection();
            mahasiswaDAO = new MahasiswaDAOImpl();
            mahasiswaDAO.setConnection(connection);
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
```

```
}

public Mahasiswa saveOrUpdate(Mahasiswa mahasiswa, boolean isUpdate){
    try {
        connection.setAutoCommit(false);
        mahasiswaDAO.saveOrUpdate(mahasiswa, isUpdate);
        connection.commit();
        connection.setAutoCommit(true);
    } catch (SQLException ex) {
        try{
            connection.rollback();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
    return mahasiswa;
}

public Mahasiswa delete(Mahasiswa mahasiswa){
    try {
        connection.setAutoCommit(false);
        mahasiswaDAO.delete(mahasiswa);
        connection.commit();
        connection.setAutoCommit(true);
    } catch (SQLException ex) {
        try{
            connection.rollback();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
    return mahasiswa;
}

public Mahasiswa getPerson(String nim){
    Mahasiswa mahasiswa = null;
    try {
        mahasiswa = mahasiswaDAO.getById(nim);
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return mahasiswa;
}

public List<Mahasiswa> getPersons(){
```

```
        try{
            return mahasiswaDAO.getAll();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return new ArrayList<Mahasiswa>();
    }

}
```

### Data Source

Kita buat satu buah class untuk mendefinisikan datasource yang akan kita gunakan. Class ini akan membuat aplikasi kita menjadi fleksibel, ketika ada perubahan database kita cukup ganti konfigurasinya dari class ini.

```
import java.sql.SQLException;

import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

public class DatabaseDataSource {

    private MysqlDataSource mysqlDataSource;

    public DatabaseDataSource() {
        mysqlDataSource = new MysqlDataSource();
    }nohp

    public MysqlDataSource getMysqlDataSource() {
        mysqlDataSource.setUser("root");
        mysqlDataSource.setPassword("root");
        mysqlDataSource.setDatabaseName("sinau");
        mysqlDataSource.setServerName("localhost");
        mysqlDataSource.setPortNumber(3306);

        return mysqlDataSource;
    }

    public void closeMysqlConnection() {
        try {
            mysqlDataSource.getConnection().close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

### Main Class

Sekarang saatnya kita panggil dari main class. Ingat yang perlu kita panggil adalah class service bukan DAO. Kalo kita gambarkan strukturnya seperti dibawah ini.

| Main |
|---|
| Service |
| DAO |
| Database |

```
import javaadvanced.session1.datasource.DatabaseDataSource;
import javaadvanced.session1.model.Mahasiswa;
import javaadvanced.session1.service.MahasiswaService;

public class MainMahasiswa {

    public static void main(String[] args) {
        DatabaseDataSource mysql = new DatabaseDataSource();

        MahasiswaService mahasiswaService = new MahasiswaService();
        mahasiswaService.setDataSource(mysql.getMysqlDataSource());

        Mahasiswa m = new Mahasiswa();
        m.setNim("12345678");
        m.setNama("Joni");
        m.setNohp("08738272726");

        mahasiswaService.saveOrUpdate(m, false);

        mysql.closeMysqlConnection();
    }
}
```

### Tugas

Buatlah menggunakan DAO untuk insert, select, update, delete pada data dosen. Field-field pada tabel dosen: nomor induk, nama, alamat, email.