

## Pengenalan Data Access Object di Java

Eko Kurniawan Khannedy

echo.khannedy@gmail.com

<http://eecchho.wordpress.com/>

### Lisensi Dokumen:

Copyright © 2003-2009 IlmuKomputer.Com

Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarluaskan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Data Access Object atau lebih terkenal singkatannya yaitu DAO merupakan Design Pattern yang biasa digunakan oleh seorang Java Developer dalam membangun sebuah Sistem berbasis Database.

DAO merupakan sebuah konsep dimana digunakan untuk menangani kasus yang terjadi dalam Busnis Logic, atau lebih gampangnya proses yang berhubungan dengan Manipulasi Data dalam Database. DAO merupakan pola membangun sebuah bisnis logic secara terstruktur sesuai dengan Entitas yang terdapat pada Database.

Misal saja, saya memiliki sebuah perangkat lunak, dimana dalam perangkat lunak itu terdapat banyak entitas, misal saja entitas Administrator, Customer, Distributor dan lain-lain. Berdasarkan konsep DAO, semua entitas tersebut harus memiliki DAO masing-masing, dimana biasanya DAO tersebut menangani proses manipulasi entitas tersebut masing-masing

Misal untuk entitas Administrator, maka harus dibuat sebuah DAO dengan nama AdministratorDao. Untuk Customer dibuat sebuah DAO dengan nama CustomerDao, dan entitas-entitas yang lainnya.

Biasanya dalam DAO tersebut terdapat fungsi-fungsi untuk memanipulasi data seperti INSERT, UPDATE, DELETE, SELECT, tergantung kebutuhan setiap entitasnya. Sehingga tidak harus semua proses DML (Data Manipulation) harus dibuat, misal saja jika Administrator tak dapat di hapus, maka tak perlu ada proses DELETE di AdministratorDao.

---

### IMPLEMENTASI DALAM JDBC (JAVA(TM) DATABASE CONNECTIVITY)

---

sekarang, saya akan bahas tentang implementasi konsep dao pada jdbc (java database connectivity).

agar lebih afdol sebaiknya pada praktek konsep DAO kali ini, saya ambil contoh sebuah kasus, yaitu kita akan membuat sebuah sistem informasi universitas, dimana pada sistem informasi tersebut memiliki sebuah entitas yang bernama Mahasiswa, Jurusan dan Fakultas. dan sekarang tugas kita adalah menerapkan konsep DAO tersebut pada sistem informasi tersebut.

sebelum masuk ke java, kita buat dulu table Mahasiswa, Fakultas dan Jurusan. saya menggunakan MySQL sebagai database-nya, jadi seperti ini struktur tabel nya :

```
create database dao_jdbc;

create table table_fakultas(
    id_fakultas bigint(20) unsigned primary key auto_increment,
    nama_fakultas varchar(45) not null
)engine=innodb;

create table table_jurusan(
    id_jurusan bigint(20) unsigned primary key auto_increment,
    nama_jurusan varchar(45) not null,
    id_fakultas bigint(20) unsigned not null
)engine=innodb;

create table table_mahasiswa(
    nim varchar(8) primary key,
    nama_depan varchar(45) not null,
    nama_belakang varchar(45),
    tanggal_lahir date not null,
    id_jurusan bigint(20) unsigned
)engine=innodb;

alter table table_jurusan add constraint fk_fakultas_jurusan
foreign key (id_fakultas) references table_fakultas(id_fakultas)
on update cascade on delete restrict;

alter table table_mahasiswa add constraint fk_jurusan_mahasiswa
foreign key (id_jurusan) references table_jurusan(id_jurusan)
on update cascade on delete restrict;
```

sekarang kita buat kelas-kelas entity nya tersebut, tapi supaya gak terlalu panjang, saya buat entitas yang Fakultas saja, karena cara membuat entity dan DAO untuk entitas-entitas yang lain hampir sama:

```
package echo.khannedy.daojdbc.entity;

public class Fakultas {

    private Long id;
    private String nama;

    public Long getId() {
        return id;
    }
```

```
}

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

public String getNama() {
    return nama;
}

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

sekarang kita sudah membuat semua kelas entitas yang telah dibutuhkan. sekarang tinggal kita terapkan konsep DAO.

dalam menerapkan konsep DAO, perlu diperhatikan bahwa biasanya DAO merupakan Interface setelah itu diimplementasi menjadi DAOImpl (Dao Implement). kenapa harus interface? kenapa tidak langsung menggunakan kelas?

alasan pertama adalah agar seluruh kode DAO bersifat HIDDEN. maksudnya? jika menggunakan interface, artinya seluruh metode dibuat tanpa adanya deklarasi secara langsung pada interface tersebut. sehingga user yang tidak berwenang tidak bisa melihat kode cara memanipulasi entitas.

lantas kalo DAO nya interface bagaimana cara membuat object dari DAO tersebut? caranya menggunakan Dependency Injection lewat Factory Method.

????????

hehehehe, jadi seperti ini cara membuatnya :

```
SampleDao dao = DaoFactory.getSampleDao();
```

bukan :

```
SampleDap dao = new SampleDaoImpl();
```

jadi harus kita buat juga sebuah kelas DaoFactory atau terserah anda ingin menamakan apa kelas tersebut :D. sekarang kita buat interface-interface DAO nya :

```
package echo.khannedy.daojdbc.dao;

import echo.khannedy.daojdbc.entity.Fakultas;
import java.sql.SQLException;
import java.util.List;

public interface FakultasDao {
```

```
void insert(Fakultas fakultas) throws SQLException;
void update(Fakultas fakultas) throws SQLException;
void delete(Fakultas fakultas) throws SQLException;
Fakultas getById(Long id) throws SQLException;
List<Fakultas> getAll() throws SQLException;
}
```

setelah selesai membuat interface-interface DAO, saatnya kita buat implementasi dari tiap interface-interface tersebut :D

```
package echo.khannedy.dao.jdbc.dao.impl;

import echo.khannedy.dao.jdbc.dao.FakultasDao;
import echo.khannedy.dao.jdbc.entity.Fakultas;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.ArrayList;
import java.util.List;

public class FakultasDaoImpl implements FakultasDao {

    private final String SQL_INSERT = "insert into table_fakultas
(id_fakultas, nama_fakultas) values (?,?)";
    private final String SQL_UPDATE = "update table_fakultas set
nama_fakultas = ? where id_fakultas = ?";
    private final String SQL_DELETE = "delete from table_fakultas where
id_fakultas = ?";
    private final String SQL_GETBYID = "select * from table_fakultas where
id_fakultas = ?";
    private final String SQL_GETALL = "select * from table_fakultas";
    private Connection connection;

    public FakultasDaoImpl(Connection connection) {
        this.connection = connection;
    }

    @Override
    public void insert(Fakultas fakultas) throws SQLException {
        PreparedStatement statement = null;
        ResultSet result = null;
        try {
            connection.setAutoCommit(false);

            statement = connection.prepareStatement(SQL_INSERT);
            statement.setNull(1, Types.BIGINT);
            statement.setString(2, fakultas.getNamaFakultas());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
        statement.setString(2, fakultas.getNama());
        statement.executeUpdate();

        result = statement.getGeneratedKeys();
        if (result.next()) {
            fakultas.setId(result.getLong(1));
        }

        connection.commit();
    } catch (SQLException exception) {
        connection.rollback();
        throw exception;
    } finally {
        try {
            connection.setAutoCommit(true);
            if (result != null) {
                result.close();
            }
            if (statement != null) {
                statement.close();
            }
        } catch (SQLException exception) {
            throw exception;
        }
    }
}

@Override
public void update(Fakultas fakultas) throws SQLException {
    PreparedStatement statement = null;
    try {
        connection.setAutoCommit(false);

        statement = connection.prepareStatement(SQL_UPDATE);
        statement.setString(1, fakultas.getNama());
        statement.setLong(2, fakultas.getId());
        statement.executeUpdate();

        connection.commit();
    } catch (SQLException exception) {
        throw exception;
    } finally {
        try {
            connection.setAutoCommit(true);
            if (statement != null) {
                statement.close();
            }
        } catch (SQLException exception) {
            throw exception;
        }
    }
}
```

```
@Override
public void delete(Fakultas fakultas) throws SQLException {
    PreparedStatement statement = null;
    try {
        connection.setAutoCommit(false);

        statement = connection.prepareStatement(SQL_DELETE);
        statement.setLong(1, fakultas.getId());
        statement.executeUpdate();

        connection.commit();
    } catch (SQLException exception) {
        throw exception;
    } finally {
        try {
            connection.setAutoCommit(true);
            if (statement != null) {
                statement.close();
            }
        } catch (SQLException exception) {
            throw exception;
        }
    }
}

@Override
public Fakultas getById(Long id) throws SQLException {
    PreparedStatement statement = null;
    ResultSet result = null;
    try {
        connection.setAutoCommit(false);

        statement = connection.prepareStatement(SQL_GETBYID);
        statement.setLong(1, id);

        result = statement.executeQuery();

        Fakultas fakultas = null;
        if (result.next()) {
            fakultas = new Fakultas();
            fakultas.setId(result.getLong("id_fakultas"));
            fakultas.setNama(result.getString("nama_fakultas"));
        }

        connection.commit();
        return fakultas;
    } catch (SQLException exception) {
        throw exception;
    } finally {
        try {
            connection.setAutoCommit(true);
        }
    }
}
```

```
        if (result != null) {
            result.close();
        }
        if (statement != null) {
            statement.close();
        }
    } catch (SQLException exception) {
        throw exception;
    }
}

@Override
public List<Fakultas> getAll() throws SQLException {
    PreparedStatement statement = null;
    ResultSet result = null;
    try {
        connection.setAutoCommit(false);

        statement = connection.prepareStatement(SQL_GETALL);

        result = statement.executeQuery();
        List<Fakultas> list = new ArrayList<Fakultas>();
        while (result.next()) {
            Fakultas fakultas = new Fakultas();
            fakultas.setId(result.getLong("id_fakultas"));
            fakultas.setNama(result.getString("nama_fakultas"));
            list.add(fakultas);
        }

        connection.commit();
        return list;
    } catch (SQLException exception) {
        throw exception;
    } finally {
        try {
            connection.setAutoCommit(true);
            if (result != null) {
                result.close();
            }
            if (statement != null) {
                statement.close();
            }
        } catch (SQLException exception) {
            throw exception;
        }
    }
}
}
```

setelah itu kita buat kelas DaoFactory untuk dijadikan kelas factory method untuk membaut DAO fakultas

```
package echo.khannedy.daojdbc.factory;

import com.mysql.jdbc.Driver;
import echo.khannedy.daojdbc.dao.FakultasDao;
import echo.khannedy.daojdbc.dao.impl.FakultasDaoImpl;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DaoFactory {

    private static Connection connection;
    private static FakultasDao fakultasDao;

    private static Connection getConnection() throws SQLException {
        if (connection == null) {
            DriverManager.registerDriver(new Driver());
            String url = "jdbc:mysql://localhost:3306/dao_jdbc";
            String user = "user";
            String password = "password";
            connection = DriverManager.getConnection(url, user,
password);
        }
        return connection;
    }

    public static FakultasDao getFakultasDao() throws SQLException {
        if (fakultasDao == null) {
            fakultasDao = new FakultasDaoImpl(getConnection());
        }
        return fakultasDao;
    }
}
```

sekarang kita telah selesai menerapkan konsep DAO pada sistem yang kita buat, nah bagaimana cara menggunakan DAO tersebut ?, dibawah ini adalah contoh sintak penggunaan DAO

```
package echo.khannedy.daojdbc;

import echo.khannedy.daojdbc.dao.FakultasDao;
import echo.khannedy.daojdbc.entity.Fakultas;
import echo.khannedy.daojdbc.factory.DaoFactory;
import java.sql.SQLException;

public class Main {

    public static void main(String[] args) throws SQLException {
        // membuat fakultas baru
        Fakultas fakultas = new Fakultas();
    }
}
```

```
fakultas.setNama("Teknik dan Ilmu Komputer");

// membuat fakultas dao
FakultasDao dao = DaoFactory.getFakultasDao();

// insert Fakultas
dao.insert(fakultas);

// mendapatkan id
Long id = fakultas.getId();

// load fakultas dari database
fakultas = dao.getById(id);

// mengubah data fakultas
fakultas.setNama("Bahasa");

// menyimpan data yang telah berubah
dao.update(fakultas);

// menghapus fakultas
dao.delete(fakultas);
}

=====
```

## IMPLEMENTASI DALAM JPA (JAVA PERSISTENCE API)

sekarang, saya akan bahas tentang implementasi konsep dao pada java persistence api

jika anda belum mengerti tentang jpa, ada baiknya anda cari info dulu tentang jpa atau belajar jpa terlebih dahulu. saya tidak akan membahas tentang jpa pada artikel ini, yang saya bahas hanya implementasi data access object dalam java persistence api, sehingga saya anggap anda mengerti jpa jika membaca artikel ini

pada artikel ini saya akan menggunakan kasus sebelumnya pada implementasi jdbc, yaitu membuat sistem informasi universitas, dimana contohnya hanya membuat dao untuk entitas fakultas. dan seperti pada artikel sebelumnya, untuk memulainya kita awali dengan membuat kelas entitas

```
package echo.khannedy.daopersistence.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "table_fakultas")
public class Fakultas implements Serializable {

    private static final long serialVersionUID = -7393147398007771743L;

    @Id
    @Column(name = "id_fakultas")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "nama_fakultas", nullable = false, length = 45)
    private String nama;

    public Long getId() {
        return id;
    }

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

    public String getNama() {
        return nama;
    }

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

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Fakultas other = (Fakultas) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 97 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

}

setelah itu karena kita menggunakan jpa, jadi kita buat dulu konfigurasi jpa pada file persistence.xml, misal saya menggunakan TopLink, sehingga konfigurasinya seperti dibawah ini :

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
  java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="persistence" transaction-type="RESOURCE_LOCAL">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>echo.khannedy.daopersistence.entity.Fakultas</class>
    <properties>
      <property name="toplink.jdbc.user" value="username"/>
      <property name="toplink.jdbc.password" value="password"/>
      <property name="toplink.jdbc.url" value="jdbc:mysql://
localhost:3306/database"/>
      <property name="toplink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
      <property name="toplink.ddl-generation" value="drop-and-create-
tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

setelah selesai membuat konfigurasi persistence, saatnya kita membuat interface untuk FakultasDao :

```
package echo.khannedy.daopersistence.dao;

import echo.khannedy.daopersistence.entity.Fakultas;
import java.util.List;

public interface FakultasDao {

  void insert(Fakultas fakultas);

  void update(Fakultas fakultas);

  void delete(Fakultas fakultas);

  Fakultas getById(Long id);

  List<Fakultas> getAll();
}
```

setelah itu buat kelas dao implementasi menggunakan java persistence api :

```
package echo.khannedy.daopersistence.impl;

import echo.khannedy.daopersistence.dao.FakultasDao;

Komunitas eLearning IlmuKomputer.Com
Copyright © 2003-2009 IlmuKomputer.Com
```

```
import echo.khannedy.daopersistence.entity.Fakultas;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceException;

public class FakultasDaoImpl implements FakultasDao {

    private EntityManagerFactory factory;

    public FakultasDaoImpl(EntityManagerFactory factory) {
        this.factory = factory;
    }

    public void insert(Fakultas fakultas) {
        EntityManager manager = factory.createEntityManager();
        try {
            manager.getTransaction().begin();
            manager.persist(fakultas);
            manager.getTransaction().commit();
        } catch (PersistenceException exception) {
            manager.getTransaction().rollback();
            throw exception;
        } finally {
            manager.close();
        }
    }

    public void update(Fakultas fakultas) {
        EntityManager manager = factory.createEntityManager();
        try {
            manager.getTransaction().begin();
            manager.merge(fakultas);
            manager.getTransaction().commit();
        } catch (PersistenceException exception) {
            manager.getTransaction().rollback();
            throw exception;
        } finally {
            manager.close();
        }
    }

    public void delete(Fakultas fakultas) {
        EntityManager manager = factory.createEntityManager();
        try {
            manager.getTransaction().begin();
            fakultas = manager
                .<Fakultas> find(Fakultas.class, fakultas.getId
());
            manager.remove(fakultas);
            manager.getTransaction().commit();
        }
```

```
        } catch (PersistenceException exception) {
            manager.getTransaction().rollback();
            throw exception;
        } finally {
            manager.close();
        }
    }

    public Fakultas getById(Long id) {
        EntityManager manager = factory.createEntityManager();
        try {
            manager.getTransaction().begin();
            Fakultas fakultas = manager.<Fakultas> find(Fakultas.class,
id);
            manager.getTransaction().commit();
            return fakultas;
        } catch (PersistenceException exception) {
            manager.getTransaction().rollback();
            throw exception;
        } finally {
            manager.close();
        }
    }

    @SuppressWarnings("unchecked")
    public List<Fakultas> getAll() {
        EntityManager manager = factory.createEntityManager();
        try {
            manager.getTransaction().begin();
            List<Fakultas> list = manager.createQuery(
                "select a from Fakultas a").getResultList();
            manager.getTransaction().commit();
            return list;
        } catch (PersistenceException exception) {
            manager.getTransaction().rollback();
            throw exception;
        } finally {
            manager.close();
        }
    }
}
```

setelah selesai membuat implementasi interface DAO, saatnya kita buat DAOFactory, dimana kali ini saya membuat kelas tersebut dengan nama PersistenceUtil :

```
package echo.khannedy.daopersistence.util;

import echo.khannedy.daopersistence.dao.FakultasDao;
import echo.khannedy.daopersistence.impl.FakultasDaoImpl;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```

```
public final class PersistenceUtil {  
  
    private static final EntityManagerFactory factory;  
    private static final FakultasDao fakultasDao;  
  
    static {  
        factory = Persistence.createEntityManagerFactory("persistence");  
        fakultasDao = new FakultasDaoImpl(factory);  
    }  
  
    public static EntityManagerFactory getFactory() {  
        return PersistenceUtil.factory;  
    }  
  
    public static FakultasDao getFakultasDao() {  
        return PersistenceUtil.fakultasDao;  
    }  
}
```

sekarang telah selesai, tinggal kita buat contoh penggunaan DAO tersebut, contohnya seperti terlihat pada sourcecode dibawah ini :

```
package echo.khannedy.daopersistence;  
  
import echo.khannedy.daopersistence.dao.FakultasDao;  
import echo.khannedy.daopersistence.entity.Fakultas;  
import echo.khannedy.daopersistence.util.PersistenceUtil;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        // membuat dao  
        FakultasDao dao = PersistenceUtil.getFakultasDao();  
  
        // membuat fakultas  
        Fakultas fakultas = new Fakultas();  
        fakultas.setNama("Teknik dan Ilmu Komputer");  
  
        // menyimpan fakultas  
        dao.insert(fakultas);  
  
        // load data fakultas  
        fakultas = dao.getById(fakultas.getId());  
  
        // menubah nama fakultas  
        fakultas.setNama("Sastra");  
  
        // menyimpan hasil pengubahan  
        dao.update(fakultas);  
    }  
}
```

```
// menghapus fakultas  
dao.delete(fakultas);  
  
}  
}
```

---

## IMPLEMENTASI DALAM HIBERNATE ANNOTATIONS

---

sekarang, saya akan bahas tentang implementasi dao pada hibernate annotation. kenapa hibernate annotation? kenapa tidak hibernate core saja?

alasa utama saya menggunakan hibernate annotation dibandingkan dengan hibernate core adalah karena menggunakan @annotation yang menurut saya lebih mudah dan lebih elegan dibandingkan kita harus membuat file XML untuk mapping kelas dan tabel

ok, kita langsung saja masuk ke inti masalah. dan masalahnya sama seperti masalah sebelum-sebelumnya yaitu membuat sistem informasi universitas, dan kalo ini menggunakan hibernate annotation. dan seperti biasa pula, saya hanya akan membahas tentang dao untuk entitas fakultas, sedangkan untuk entitas lainnya bisa anda buat sendiri :D

sekarang tahap pertama adalah kita buat kelas Fakultas terlebih dahulu :

```
package echo.khannedy.daohibernate.entity;  
  
import java.io.Serializable;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "table_fakultas")  
public class Fakultas implements Serializable {  
  
    private static final long serialVersionUID = -8042959022405932340L;  
  
    @Id  
    @Column(name = "id_fakultas")  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    @Column(name = "nama_fakultas", nullable = false, length = 45)  
    private String nama;  
  
    public Long getId() {  
        return id;  
    }
```

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

public String getNama() {
    return nama;
}

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

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Fakultas other = (Fakultas) obj;
    if (this.id != other.id)
        && (this.id == null || !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 79 * hash + (this.id != null ? this.id.hashCode() : 0);
    return hash;
}
}
```

setelah membuat kelas entitas, kita buat konfigurasi hibernate annotation menggunakan file hibernate.cfg.xml yang disimpan pada root package :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/database</property>
```

```
<property name="hibernate.connection.username">username</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.hbm2ddl.auto">create-drop</property>
<mapping class="echo.khannedy.daohibernate.entity.Fakultas" />
</session-factory>
</hibernate-configuration>
```

sekarang saatnya kita buat interface dao untuk kelas entity fakultas diatas :

```
package echo.khannedy.daohibernate.dao;

import echo.khannedy.daohibernate.entity.Fakultas;
import java.util.List;

public interface FakultasDao {

    void insert(Fakultas fakultas);

    void update(Fakultas fakultas);

    void delete(Fakultas fakultas);

    Fakultas getById(Long id);

    List<Fakultas> getAll();
}
```

setelah membuat interface fakultas dao, saatnya kita buas kelas implementasi dari interface fakultas dao diatas :

```
package echo.khannedy.daohibernate.impl;

import echo.khannedy.daohibernate.dao.FakultasDao;
import echo.khannedy.daohibernate.entity.Fakultas;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

public class FakultasDaoImpl implements FakultasDao {

    private SessionFactory factory;

    public FakultasDaoImpl(SessionFactory factory) {
        this.factory = factory;
    }

    public void insert(Fakultas fakultas) {
        Session session = factory.openSession();
        try {
            session.beginTransaction();
            session.save(fakultas);
        }
```

```
        session.getTransaction().commit();
    } catch (HibernateException exception) {
        session.getTransaction().rollback();
        throw exception;
    } finally {
        session.close();
    }
}

public void update(Fakultas fakultas) {
    Session session = factory.openSession();
    try {
        session.beginTransaction();
        session.update(fakultas);
        session.getTransaction().commit();
    } catch (HibernateException exception) {
        session.getTransaction().rollback();
        throw exception;
    } finally {
        session.close();
    }
}

public void delete(Fakultas fakultas) {
    Session session = factory.openSession();
    try {
        session.beginTransaction();
        session.delete(fakultas);
        session.getTransaction().commit();
    } catch (HibernateException exception) {
        session.getTransaction().rollback();
        throw exception;
    } finally {
        session.close();
    }
}

public Fakultas getById(Long id) {
    Session session = factory.openSession();
    try {
        session.beginTransaction();
        Fakultas fakultas = (Fakultas) session.get(Fakultas.class,
id);
        session.getTransaction().commit();
        return fakultas;
    } catch (HibernateException exception) {
        session.getTransaction().rollback();
        throw exception;
    } finally {
        session.close();
    }
}
```

```
@SuppressWarnings("unchecked")
public List<Fakultas> getAll() {
    Session session = factory.openSession();
    try {
        session.beginTransaction();
        List<Fakultas> list = session.createCriteria(Fakultas.
class).list();
        session.getTransaction().commit();
        return list;
    } catch (HibernateException exception) {
        session.getTransaction().rollback();
        throw exception;
    } finally {
        session.close();
    }
}
```

sebenarnya hampir sama cara pembuatan dao di hibernate annotation dan java persistence api, yang membedakan adalah hibernate menggunakan Session dan SessionFactory sedangkan java persistence api menggunakan EntityManager dan EntityManagerFactory

sekarang kita buat kelas dao factory, dimana pada artikel ini saya ubah namanya menjadi HibernateUtil :

```
package echo.khannedy.daohibernate.util;

import echo.khannedy.daohibernate.dao.FakultasDao;
import echo.khannedy.daohibernate.impl.FakultasDaoImpl;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;
    private static final FakultasDao fakultasDao;

    static {
        sessionFactory = new AnnotationConfiguration().configure()
            .buildSessionFactory();
        fakultasDao = new FakultasDaoImpl(sessionFactory);
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static FakultasDao getFakultasDao() {
        return fakultasDao;
    }
}
```

```
}
```

selesai sudah implementasi dao pada hibernate, saatnya kita uji coba :

```
package echo.khannedy.daohibernate;

import echo.khannedy.daohibernate.dao.FakultasDao;
import echo.khannedy.daohibernate.entity.Fakultas;
import echo.khannedy.daohibernate.util.HibernateUtil;

public class Main {

    public static void main(String[] args) {
        // membuat dao
        FakultasDao dao = HibernateUtil.getFakultasDao();

        // membuat fakultas
        Fakultas fakultas = new Fakultas();
        fakultas.setNama("Teknik dan Ilmu Komputer");

        // menyimpan fakultas
        dao.insert(fakultas);

        // meload fakultas
        fakultas = dao.getById(fakultas.getId());

        // menubah nama fakultas
        fakultas.setNama("Sastra");

        // menyimpan hasil ke database
        dao.update(fakultas);

        // menghapus dari database
        dao.delete(fakultas);
    }
}
```

---

## KESALAHAN DALAM MENERAPKAN KONSEP DATA ACCESS OBJECT

---

sekarang, saya akan membahas tentang kesalahan dalam membuat dao, biasanya ini sering terjadi jika seseorang baru belajar dao. maka dari itu penting sekali diperhatikan jika anda baru belajar dao

jika diperhatikan tidak ada yang salah dengan contoh-contoh dao yang pada artikel sebelumnya? yup memang tidak ada yang salah. namun dalam keadaan tertentu akan menjadi sangat fatal jika dao tersebut di gunakan. maksudnya?

coba kita perhatikan pada contoh-contoh dao pada artikel sebelumnya...

setiap metode dao merupakan metode yang bersifat TRANSAKSI, yup setiap proses pada metode tersebut merupakan proses TRANSAKSI. lantas bagaimana jika kita mendapatkan kasus yang harus menggunakan banyak proses DAO namun proses tersebut harus DAO. contohnya...

misal kita memiliki kasus sistem perpustakaan yang memiliki Buku dan Peminjaman, sehingga dibuat dao seperti dibawah ini :

```
public interface BukuDao{  
    void simpanBuku(Buku buku);  
    void ubahStatusBuku(Buku, boolean dipinjam);  
    void ...  
}  
  
public interface PeminjamanDao{  
    void simpanPeminjaman(Peminjaman peminjaman);  
    void ...  
}
```

misal pada saat peminjaman tentunya kita harus menggunakan PeminjamanDao.simpanPeminjaman(...) untuk menyimpan peminjaman dan juga mengubah status buku, bahwa buku tersebut sedang dipinjam menggunakan metode BukuDao.ubahStatusBuku(true), sehingga seperti inilah kodenya :

```
Peminjaman peminjaman = ...  
peminjaman.setTanggalPinjam(new Date());  
peminjaman.setPeminjam(anggota);  
peminjaman.setBuku(buku);  
  
// simpan peminjaman  
peminjamanDao.simpanPeminjaman(peminjaman);  
// ubah status buku menjadi dipinjam  
bukuDao.ubahStatusBuku(buku, true);
```

jika diperhatikan memang tidak ada yang salah :D tapi jika dilihat lebih teliti, ini adalah sesuai yang salah :D why? mari kita liat lebih dalam pada kedua perintah dibawah ini :

```
// simpan peminjaman  
peminjamanDao.simpanPeminjaman(peminjaman);  
// ubah status buku menjadi dipinjam  
bukuDao.ubahStatusBuku(buku, true);
```

jika dijabarkan, maka hasilnya seperti ini :

```
// simpan peminjaman  
// peminjamanDao.simpanPeminjaman(peminjaman);  
transaksi.begin();  
koneksi.simpan(peminjaman);  
transaksi.commit();  
  
// ubah status buku menjadi dipinjam  
// bukuDao.ubahStatusBuku(buku, true);  
transaksi.begin();  
koneksi.ubah(status buku);  
transaksi.commit();
```

bisa dilihat pada kode diatas terdapat 2 buat transaksi, dimana transaksi pertama adalah transaksi menyimpan penyimpanan dan transaksi kedua adalah transaksi mengubah status buku

seharusnya hanya boleh terjadi 1 transaksi pada proses peminjaman tersebut, kenapa? bayangkan jika saat menyimpan data peminjaman berhasil namun saat mengubah status buku terjadi error di database sehingga menyebabkan gagalnya proses ubah status tersebut, jika hal ini terjadi, maka proses simpan data peminjaman berhasil, namun status buku masih false (artinya buku tidak dipinjam)

mungkin hal ini sepele, namun bayangkan jika proses transaksi tersebut melibatkan transaksi E'COMMERCE atau transaksi BANK, pastinya kesalahan seperti itu akan menjadi FATAL, seharusnya proses diatas terjadi dalam 1 transaksi, artinya jika salah satu proses gagal, maka proses yang lainnya pun harus digagalkan, sedangkan proses dianggap berhasil jika semua proses berhasil

jadi seharusnya dalam satu proses, jangan sampai memanggil 2 metode DAO, sehingga pada kasus diatas sebaiknya pada proses simpanPeminjaman milik PeminjamanDao, ditambah sebuah query untuk mengubah status buku, sehingga seharusnya seperti ini :

```
public class PeminjamanDaoImpl implements PeminjamanDao{  
  
    public void simpanPeminjaman(Peminjaman peminjaman){  
        transaksi.begin();  
  
        koneksi.insert(peminjaman);  
        koneksi.ubahStatus(peminjaman.getBuku(), true);  
  
        transaksi.commit();  
    }  
  
}
```

mudah-mudahan dengan dibahasnya masalah ini, akan membuat anda terhindar dari masalah yang seperti ini. walaupun masalah ini terlihat sepele, namun ini sangat fatal jika anda membangun sistem informasi yang BESAR :D

## Tentang Penulis



Penulis bernama EKO KURNIAWAN KHANNEDY, penulis ahir di Bekasi pada tanggal 29 Desember 1988, penulis adalah siswa lulusan SDN Sukaasih (2000), SLTPN 1 Kalijati (2003) dan SMAN 1 Subang (2006). sekarang penulis adalah mahasiswa UNIKOM (UNIVERSITAS KOMPUTER INDONESIA) Bandung.

Penulis aktif menulis di :

- <http://eecchhoowordpress.com/>

Penulis juga dapat dihubungi melalui email di :

- [echo.khannedy@gmail.com](mailto:echo.khannedy@gmail.com)

atau dapat dihubungi melalui hp di :

- **085292775999**

NB dari Penulis: **HARAP SMS TERLEBIH DAHULU JIKA INGIN MENELPON**