

# Daftar ISI

[Daftar ISI 2](#_heading=h.gjdgxs)

[1.](#_heading=h.30j0zll) Java Transfer Object 3

[2.](#_heading=h.1fob9te) JPA 6

[a.](#_heading=h.3znysh7) Mengapa menggunakan JPA 6

[b.](#_heading=h.2et92p0) Keuntungan menggunakan JPA 6

[c.](#_heading=h.tyjcwt) ORM 6

[d.](#_heading=h.3dy6vkm) Langkah-langkah menggunakan JPA di Spring Boot 7

[e.](#_heading=h.1t3h5sf) JPA Entity 8

[Repository gitlab 9](#_heading=h.4d34og8)

[Daftar Pustaka 10](#_heading=h.2s8eyo1)

# Java Transfer Object

**Apa itu DTO ?**

DTO adalah singkatan untuk Data Transfer Object , sehingga digunakan untuk mentransfer data antara kelas dan modul aplikasi Anda.

* DTO seharusnya hanya berisi bidang pribadi untuk data, getter, setter, dan konstruktor Anda.
* DTO tidak disarankan untuk menambahkan metode logika bisnis ke kelas tersebut, tetapi tidak masalah untuk menambahkan beberapa metode util.

**APA itu DAO ?**

DAO adalah singkatan untuk Data Access Object , sehingga harus merangkum logika untuk mengambil, menyimpan, dan memperbarui data dalam penyimpanan data Anda (database, sistem file, apa pun).

Berikut adalah contoh tampilan antarmuka DAO dan DTO:

**Contoh implementasi bagaimana ?**

Contoh nya pada kasus Data Entitas Mahasiswa ingin Mengunakan DTO.

Lahkah nya adalah

1. Membuat class entity mahasiswa

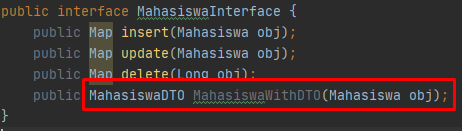
| @Data @Entity @Table(name = "mahasiswa") public class Mahasiswa implements Serializable {  @Id  @Column(name="id")  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;  @Column(name = "nama", nullable = false, length = 45)  private String nama;  @Column(name = "nim", nullable = false, length = 16)  private String nim;  @Column(name = "alamat", columnDefinition="TEXT")  private String alamat; } |
| --- |

1. Membuat Class DTO

| @Data public class MahasiswaDTO {  public MahasiswaDTO(Long id, String nama, String nim, String alamat) {  this.id = id;  this.nama = nama;  this.nim = nim;  this.alamat = alamat;  }  private Long id;   private String nama;   private String nim;   private String alamat; } |
| --- |

1. Proses DTO

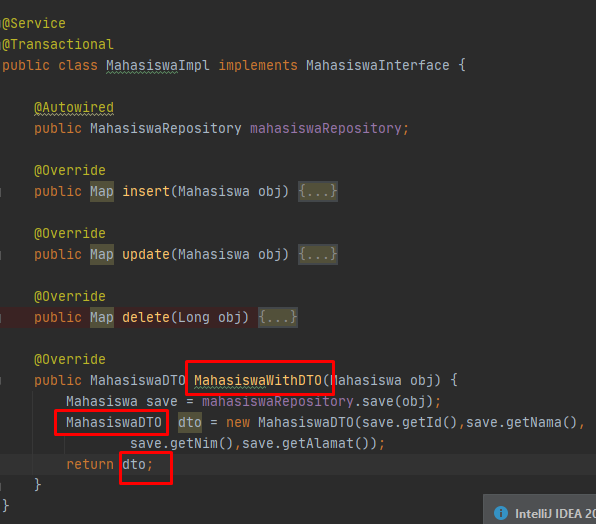
Interface



Gambar Mahasiswa DTO

Interface **MahasiswaInterface** berisikanmethod, yang nanti nya akan diimplementasikan/ override ke pada class yang mengeimplementasikanya. Class yang mengimplementasikanya ditandai dengan adanya “implement **MahasiswaInterface”** pada class tersebut. Sebagai contoh, silahkan lihatlah class berikut yang mengimplementasikan **MahasiswaInterface.**

Class Implementasi dari Interface



Gambar Implementasi DTO

Pada gambar diatas, terlihat method MahasiswaWithDTO digunakan untuk mengambil value dari Class entity Mahasiswa. Class MahasiswaDTO digunakan sebagai data transfer objek dari class mahasiswa. Itu ditandai dengan adanya return objek MahasiswaDTO dto.

Sampai sini paham ya, cara penguunaan DTO.

Biasanya DTO ini juga dipakai saat kita ingin mengabungkan beberapa data pada satu class yang sama. Biasanya juga digunakan sebagai response pada rest api.

# JPA

Apa itu JPA ?

JPA adalah sebuah spesifikasi atau standar, tentang bagaimana kita menyimpan sebuah objek di Java. JPA sendiri adalah singkatan dari Java Persistence API. *Persist* sendiri berarti: teguh; tahan lama; tidak berubah dalam kurun waktu tertentu. Umumnya objek di Java adalah data. Data biasanya disimpan agar dapat digunakan kembali (reusable). *Persistence* sendiri dimaksudkan agar objek Java yang pakai di aplikasi ini dapat disimpan sehingga tetap ada dan dapat digunakan kembali nantinya meskipun (misalnya) aplikasi yang memproduksi atau mengkonsumsi objek ini sudah mati atau tidak berjalan lagi.

Berbicara tentang JPA, biasanya tidak terlepas dari berbicara tentang database. JPA ada dalam package javax.persistence.\*.

## Mengapa menggunakan JPA

Sebagai standar untuk mengimplementasikan:

* data mana yang akan disimpan dan,
* bagaimana cara kita menyimpan data tersebut.

## Keuntungan menggunakan JPA

Dengan menggunakan JPA, kita tidak perlu ribet memikirkan bagaimana data tersebut akan di-*store* ke database, cara menerjemahkan sebuah objek menjadi tabel di database ataupun cara kita mengambil 1 row data dari tabel dan diterjemahkan kembali menjadi sebuah objek yang dapat kita gunakan langsung di dalam aplikasi Java.

## ORM

Apa hubunganya ORM dengan JPA ?

Object Relational Mapping (ORM) sederhananya adalah framework yang mengimplementasi standar yang ditetapkan oleh JPA. ORM melakukan translasi dari sebuah objek Java ke dalam tabel database yang dimaksud. Jika JPA menetapkan bahwa, agar sebuah class Java dapat ditranslasi menjadi tabel, class tersebut harus dipasangi anotasi @Entity. Lalu tugas ORM adalah menerjemahkan class tersebut ke dalam tabel. ORM juga melakukan mapping field atau properti dari sebuah class menjadi kolom pada tabel di database. Sebuah properti firstName pada class Student akan menjadi kolom first\_name pada tabe student di database, misalnya.

Framework ORM yang paling mature dan banyak digunakan untuk implementasi JPA adalah Hibernate.

## Langkah-langkah menggunakan JPA di Spring Boot

**Add depedencis Pom.xml**

| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency>  Tambahakn lib posgres : Database yang digunakan  <dependency>  <groupId>org.postgresql</groupId>  <artifactId>postgresql</artifactId>  <scope>runtime</scope> </dependency> |
| --- |

**Konfigurasi application properties**

| spring.datasource.url=jdbc:postgresql://localhost:5433/investree spring.datasource.username=postgres spring.datasource.password=12345spring.jpa.show-sql=true spring.jpa.hibernate.ddl-auto=update |
| --- |

## JPA Entity

JPA Entity adalah sebuah *class* yang merepresentasikan *table* yang ada di *database*.

***Contoh***

| @Data @Entity @Table(name = "mahasiswa") public class Mahasiswa implements Serializable {  @Id  @Column(name="id")  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;  @Column(name = "nama", nullable = false, length = 45)  private String nama;  @Column(name = "nim", nullable = false, length = 16)  private String nim;  @Column(name = "alamat", columnDefinition="TEXT")  private String alamat; } |
| --- |

Secara otomatis, nama dari *entity* akan disesuaikan dengan nama *class* tersebut. Dengan menambahkan anotasi **@Entity**tersebut, artinya *class* **Tour** kita sudah di-*mapping* dengan *table* **tour**. Berikutnya, layaknya sebuah *table*, anotasi *entity* ini juga harus memiliki sebuah *primary* *key* yang ditandai dengan anotasi **@Id**. Jika kita ingin *primary key*yang kita miliki di-*generate* secara otomatis, maka kita bisa menggunakan anotasi **@GeneratedValue**. Anotasi **@GeneratedValue** ini memiliki 4 jenis strategi untuk *generate primary key*, yakni ***AUTO, TABLE, SEQUENCE***dan***IDENTITY.***

Biasanya kita ingin men-*generate primary key* secara *auto increment*, maka kita bisa gunakan strategi **IDENTITY.** Selain itu kita juga bisa menggunakan anotasi **@Column** untuk *class* properti yang kita miliki agar lebih spesifik. Misalnya untuk sebuah *description*, kita ingin *column* tersebut bisa menampung 2000 *character*, maka kita dapat spesifikkan dengan cara :

| *@Column*(name="description", length = 2000) *public* String description; |
| --- |

# 

# Repository gitlab

**https://gitlab.com/rikialdi/investree/-/tree/dto**

# Daftar Pustaka

**https://gitlab.com/rikialdi/investree/-/tree/dto**