PEMROGRAMAN BERORIENTASI OBJEK LANJUT – BS405

PERT 5 – JAVA DATABASE API BAG. 02

BY : SENDY FERDIAN SUJADI, S.KOM., M.T., CEH, CEI, MTCNA, MTCRE, MTCINE, MTA

TODAY'S MENUS:

Object Relational Mapping with Hibernate

- Relationship Mapping
- Inheritance Mapping

RELATIONSHIP MAPPING

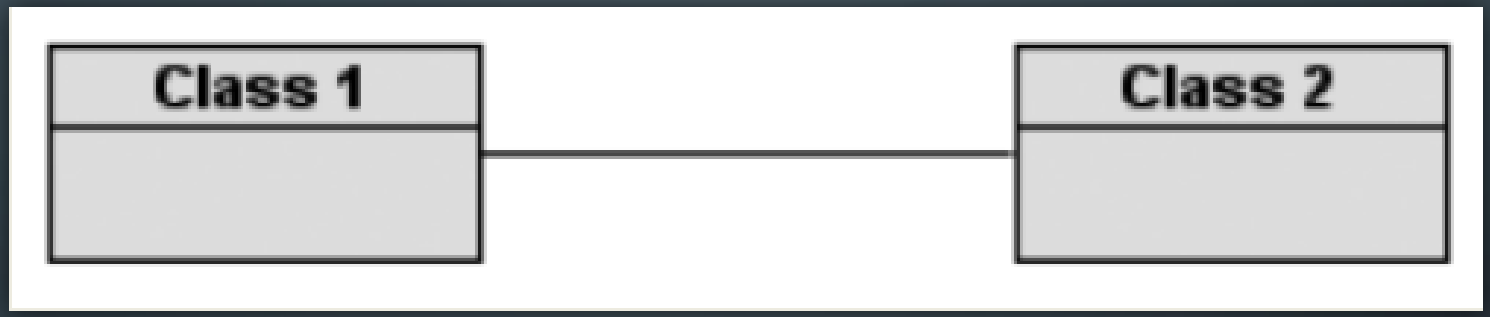
- Di dalam pemrograman berorientasi objek, kita mengenal adanya asosiasi antara class yang satu dengan class yang lainnya.
- Melalui asosiasi ini, suatu object dapat mengakibatkan object lain untuk menjalankan suatu fungsi/method.
- Ada beberapa jenis asosiasi yang terdapat antar class.
- Asosiasi ini memiliki arah: unidirectional (satu arah) atau bidirectional (dua arah)
- Kita menenal sintaks DOT (.) untuk melakukan navigasi method-method di dalam suatu object.
- Contoh: customer.getAddress().getCountry() berarti melakukan navigasi dari object Customer ke Address lalu ke Country.

ASOSIASI UNIDIRECTIONAL ANTARA DUA CLASS

- Di dalam UML, untuk merepresentasikan asosiasi unidirectional antara dua class, kita dapat menggunakan tanda panah, sbb:

ASOSIASI BIDIRECTIONAL ANTARA DUA CLASS

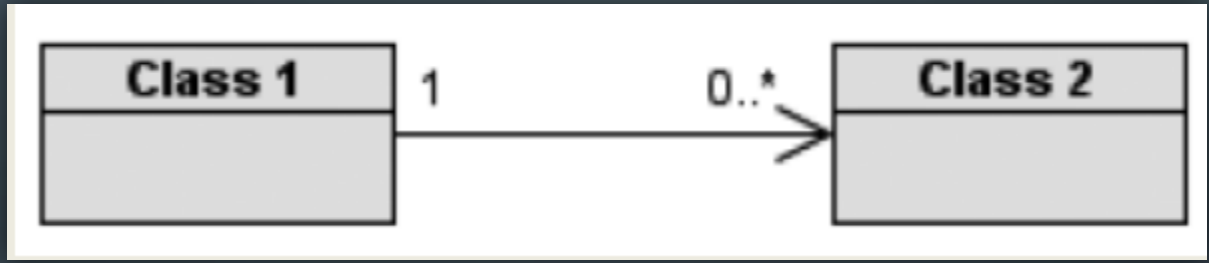
- Sedangkan untuk menunjukkan asosiasi bidirectional, kita tidak perlu menggunakan tanda panah sbb:



- Di dalam Java, hal ini berarti Class1 memiliki atribut bertipe Class2, dan Class2 memiliki atribut bertipe Class1

MULTIPLICITY PADA ASOSIASI CLASS

- Suatu asosiasi dapat juga memiliki multiplicity atau cardinality.
- Setiap ujung dari asosiasi dapat menentukan berapa banyak object yang terlibat di dalam asosiasi tsb.
- Contoh berikut: satu instance Class1 merujuk/refer pada nol atau lebih instance Class2



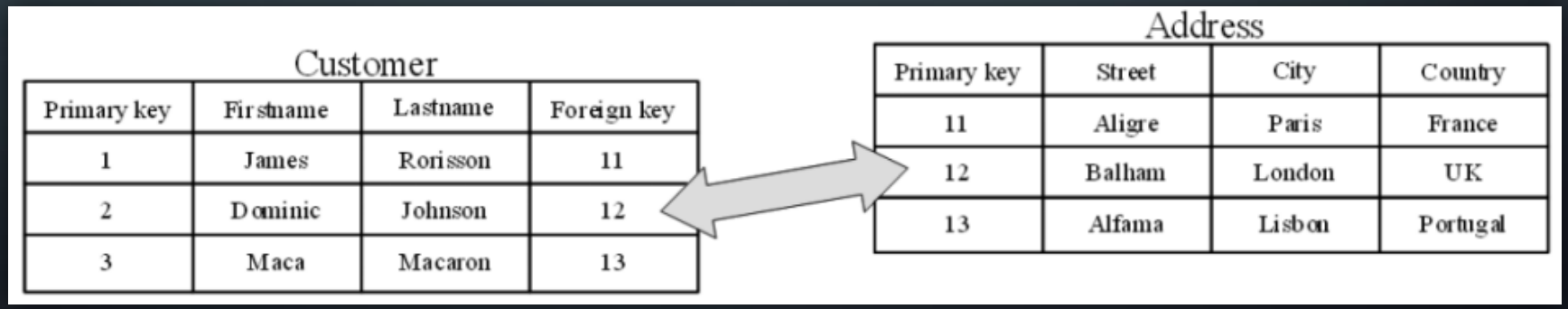
- Pada UML:
  - 0..1 berarti kita akan memiliki minimal nol object dan maksimal satu object.
  - 1 berarti kita hanya memiliki satu instance/object.
  - 1..* berarti kita hapat memiliki satu atau lebih instance/object
  - 3..6 berarti kita dapat memiliki antara 3 s/d 6 instance/object
- Pada Java, asosiasi yang menggambarkan lebih dari satu object harus menggunakan salah satu tipe data collection, yaitu: java.util.Collection, java.util.Set, java.util.List, atau java.util.Map

RELATIONSHIPS PADA RDBMS

- Berbeda dengan yang terjadi di dunia RDBMS, kita hanya mengenal adanya kumpulan antar RELATION (disebut: tabel) yang berarti apapun yang kita modelkan di dalam ERD pastilah merupakan suatu tabel.
- Untuk memodelkan asosiasi, kita tidak memiliki LIST, SET, atau MAP. Kita hanya memiliki TABEL.
- Di dalam Java, ketika kita memiliki asosiasi antara satu class dengan class lainnya, maka di dalam database nantinya kita akan memiliki TABLE REFERENCE.
- REFERENCE ini dapat dimodelkan menjadi dua cara:
  - Menggunakan FOREIGN KEY (join column)
  - Menggunakan JOIN TABLE

CARA 1
RELATIONSHIP MENGGUNAKAN JOIN COLUMN

- Sebagai contoh, misalkan seorang CUSTOMER memiliki satu ADDRESS, yang berarti relasi one-to-one.
- Di dalam Java, kita akan memiliki class Customer dengan atribut Address.
- Di dalam database, kita dapat memiliki tabel CUSTOMER yang menunjuk pada ADDRESS menggunakan kolom foreign key (join column), sbb:



Customer

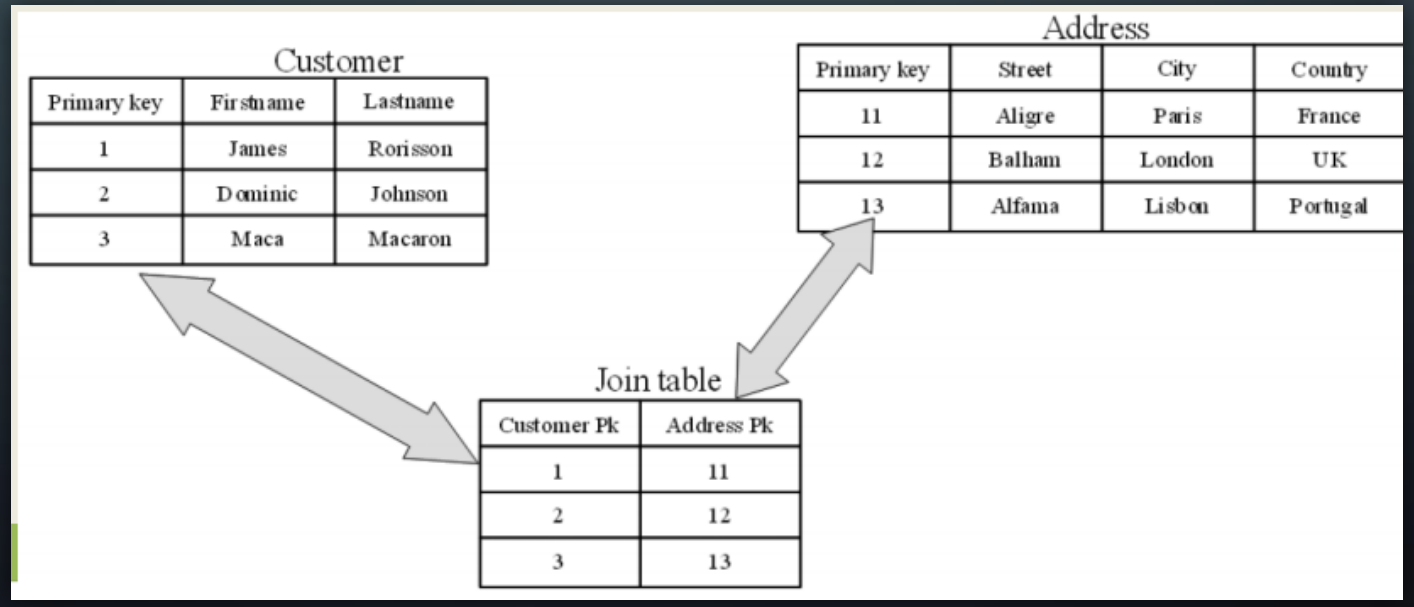
Primary key	Firstname	Lastname	Foreign key
1	James	Rorisson	11
2	Dominic	Johnson	12
3	Maca	Macaron	13

Address

Primary key	Street	City	Country
11	Aligre	Paris	France
12	Balham	London	UK
13	Alfama	Lisbon	Portugal

CARA 2: RELATIONSHIP MENGGUNAKAN JOIN TABLE

- Cara yang kedua adalah menggunakan JOIN TABLE.
- Tabel CUSTOMER tidak menyimpan foreign key dari ADDRESS lagi.
- Kita gunakan tabel tambahan yang dibuat untuk menampung informasi relationship dengan menyimpan foreign key dari kedua tabel, sbb:



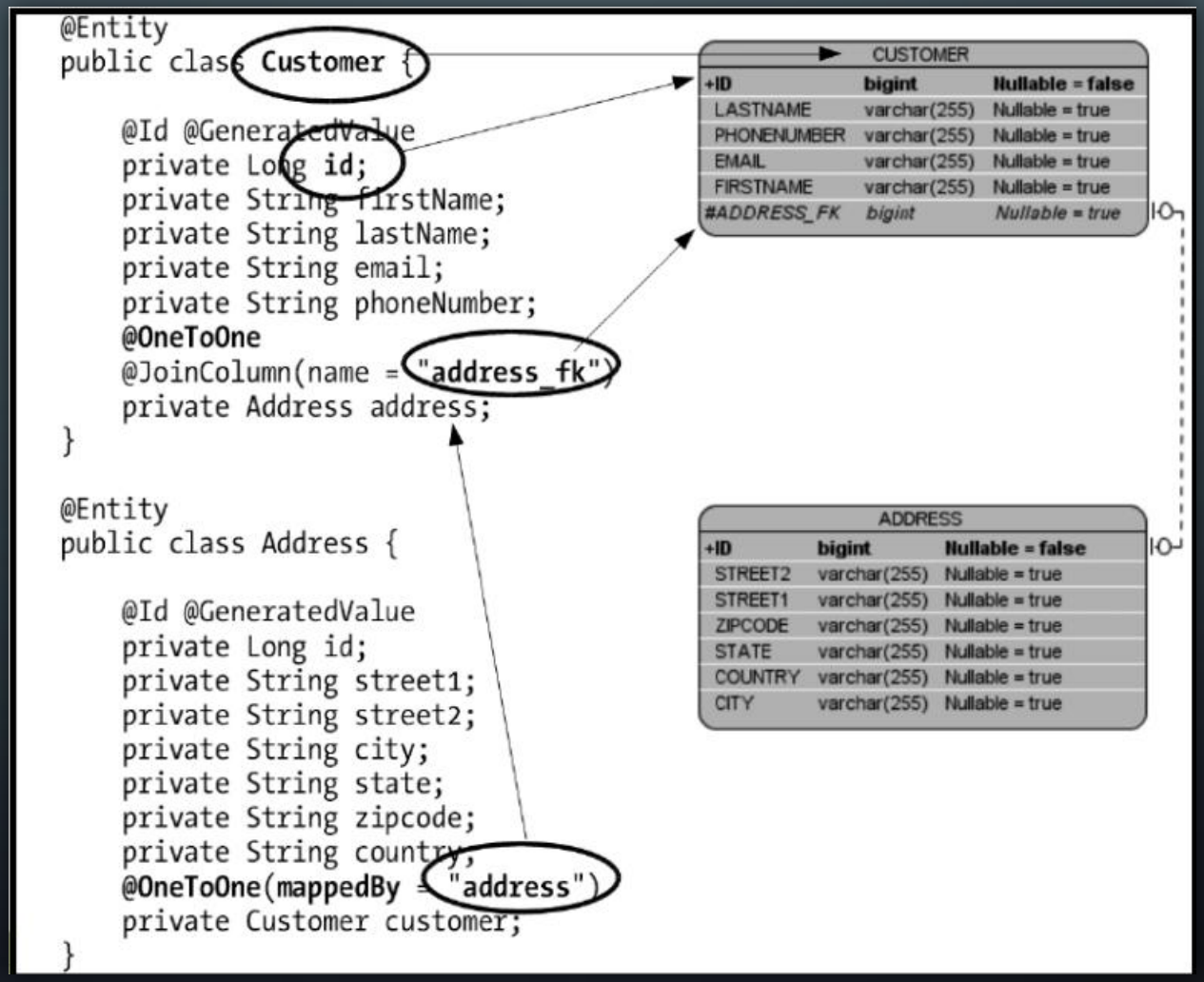
ENTITY RELATIONSHIP

- Entity Relationships dibagi menjadi 4 jenis:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Setiap anotasi tersebut dapat digunakan untuk alur unidirectional ataupun bidirectional

TEKNIK PEMETAAN DI JAVA

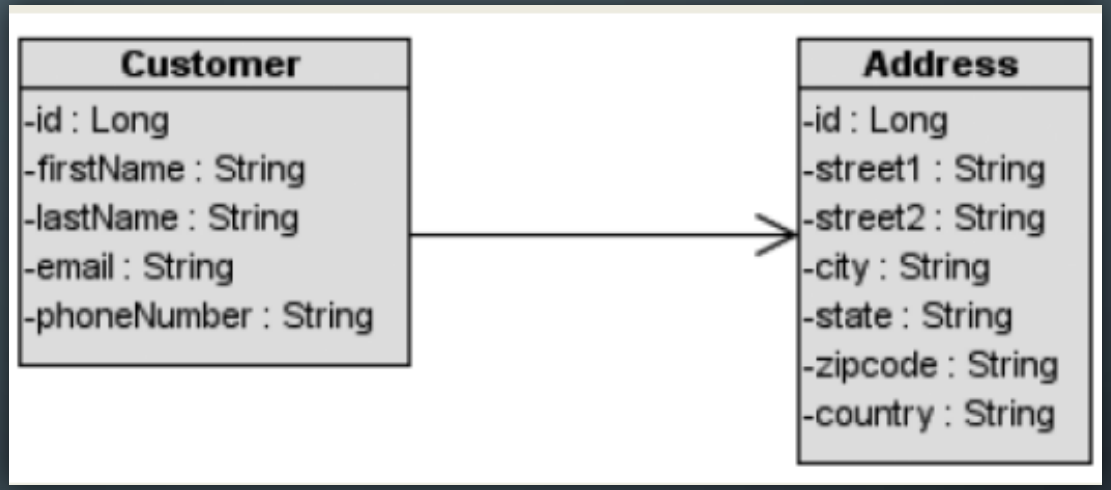
- Untuk menyatakan class manakah yang akan menjadi OWNING SIDE dan class mana yang akan menjadi INVERSE SIDE, kita harus menggunakan elemen **mappedBy** di dalam anotasi @OneToOne, @OneToMany, dan @ManyToMany.
- **mappedBy** ini akan mengidentifikasi atribut yang menjadi pemiliki relasi dan wajib digunakan untuk relasi yang bersifat bidirectional.

PEMETAAN JAVA VS DATABASE



KASUS 1:
ASOSIASI UNIDIRECTIONAL ANTARA CUSTOMER DAN ADDRESS @ONETOONE UNIDIRECTIONAL

- Pada relasi unidirectional, entitas Customer memiliki atribut bertipe Address, sbb:



- Relasi ini bersifat One-To-One dan Satu arah dari Customer ke Address.
- Customer adalah pemilik relasi (OWNER).
- Di dalam database, artinya bahwa tabel CUSTOMER akan memiliki foreign key yang mengacu pada tabel ADDRESS.
- Di dalam Java, artinya bahwa Custombер akan memiliki atribut Address.

SOURCE CODE: CUSTOMER DENGAN SATU ADDRESS

```
@Entity
public class Customer {
  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private String email;
  private String phoneNumber;
  private Address address;
  // Constructors, getters, setters
}
```

```
@Entity
public class Address {
  @Id @GeneratedValue
  private Long id;
  private String street1;
  private String street2;
  private String city;
  private String state;
  private String zipcode;
  private String country;
  // Constructors, getters, setters
}
```

```
create table CUSTOMER (
  ID BIGINT not null,
  FIRSTNAME VARCHAR(255),
  LASTNAME VARCHAR(255),
  EMAIL VARCHAR(255),
  PHONENUMBER VARCHAR(255),
  ADDRESS_ID BIGINT,
  primary key (ID),
  foreign key (ADDRESS_ID) references ADDRESS(ID)
);
```

```
create table ADDRESS (
  ID BIGINT not null,
  STREET1 VARCHAR(255),
  STREET2 VARCHAR(255),
  CITY VARCHAR(255),
  STATE VARCHAR(255),
  ZIPCODE VARCHAR(255),
  COUNTRY VARCHAR(255),
  primary key (ID)
);
```

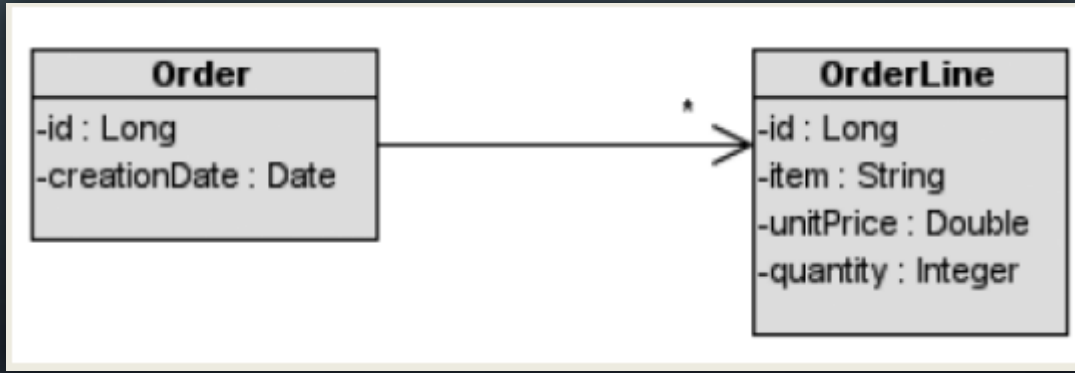
CUSTOMIZING ENTITAS CUSTOMER MENGGUNAKAN ANOTASI

- Kita dapat melakukan customizing juga terhadap mapping, dengan menggunakan dua buah anotasi yaitu @OneToOne dan @JoinColumn sbb:

```
@Entity
public class Customer {
  @Id @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private String email;
  private String phoneNumber;
  @OneToOne (fetch = FetchType.LAZY)
  @JoinColumn(name = "add_fk", nullable = false)
  private Address address;
  // Constructors, getters, setters
}
```

@ONETOMANY UNIDIRECTIONAL

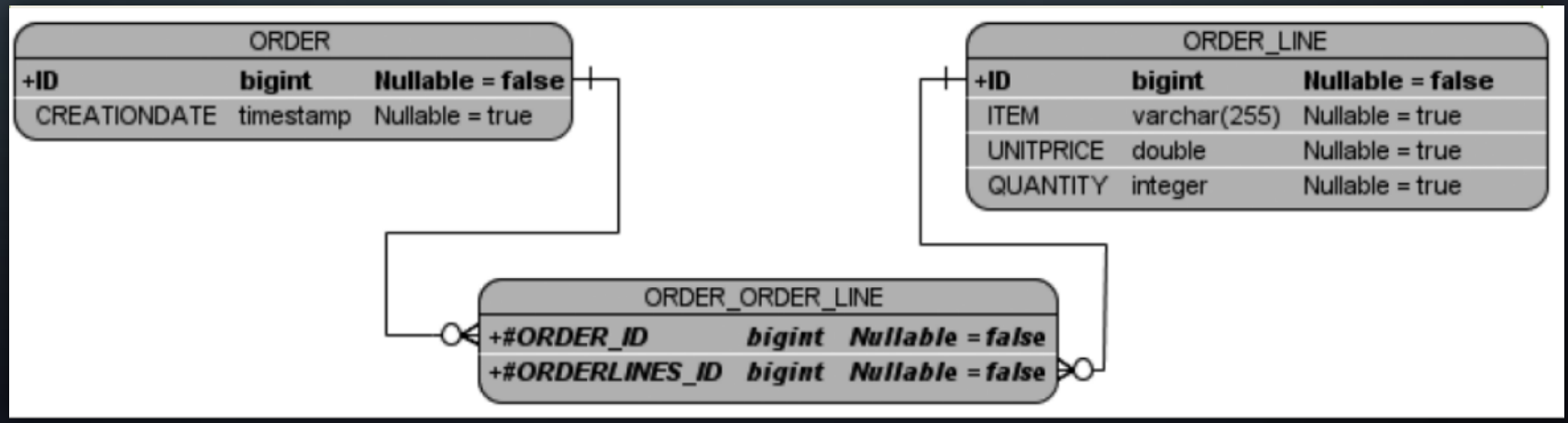
- Relasi one-to-many digunakan pada saat satu object dapat mengacu ke banyak objek yg lain
- Misalnya: sebuah purchase order dapat terdiri dari beberapa order line.
- Order adalah sisi “ONE” (source) dan OrderLine adalah sisi “Many” (target)



ENTITAS DAN TABEL ORDER BERISI ORDERLINES

```java
@Entity
public class Order {
  @Id @GeneratedValue
  private Long id;
  @Temporal(TemporalType.TIMESTAMP)
  private Date creationDate;
  private List<OrderLine> orderLines;
  // Constructors, getters, setters
}
```

```java
@Entity
@Table(name = "order_line")
public class OrderLine {
  @Id @GeneratedValue
  private Long id;
  private String item;
  private Double unitPrice;
  private Integer quantity;
  // Constructors, getters, setters
}
```



CUSTOMIZING ENTITAS ORDER MENGGUNAKAN ANOTASI JOINTABLE

```
@Entity
public class Order {
  @Id @GeneratedValue
  private Long id;
  @Temporal(TemporalType.TIMESTAMP)
  private Date creationDate;
  @OneToMany
  @JoinTable(name = "jnd_ord_line",
        joinColumns = @JoinColumn(name = "order_fk"),
        inverseJoinColumns = @JoinColumn(name = "order_line_fk") )
  private List<OrderLine> orderLines;
  // Constructors, getters, setters
}
```

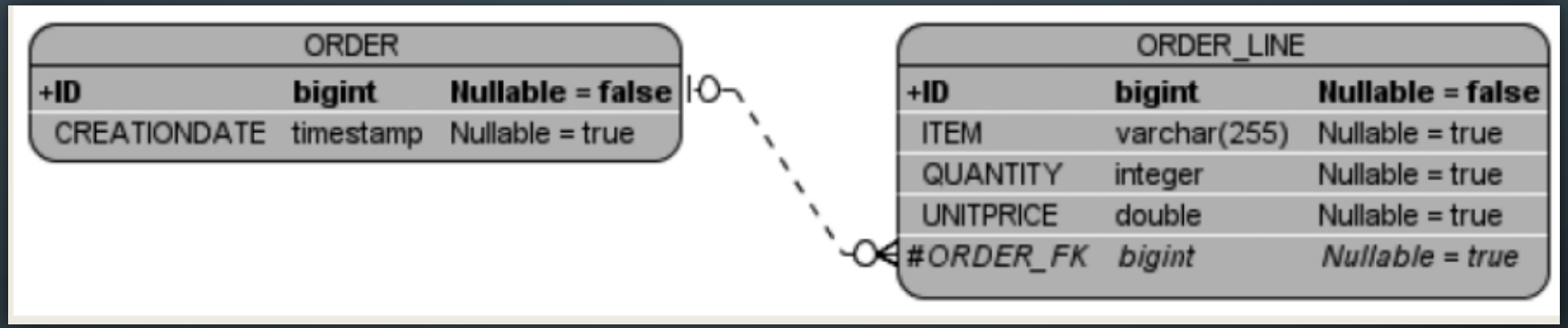
```
create table JND_ORD_LINE (
  ORDER_FK BIGINT not null,
  ORDER_LINE_FK BIGINT not null,
  primary key (ORDER_FK, ORDER_LINE_FK),
  foreign key (ORDER_LINE_FK) references
  ORDER_LINE(ID),
  foreign key (ORDER_FK) references ORDER(ID)
);
```

CUSTOMIZING ENTITAS ORDER MENGGUNAKAN ANOTASI JOINCOLUMN

- Aturan default untuk relasi unidirectional one-to-many adalah dengan menggunakan JOIN TABLE, tetapi kita dapat mengubahnya menjadi menggunakan FOREIGN KEY dengan menggunakan anotasi @JoinColumn, sbb:

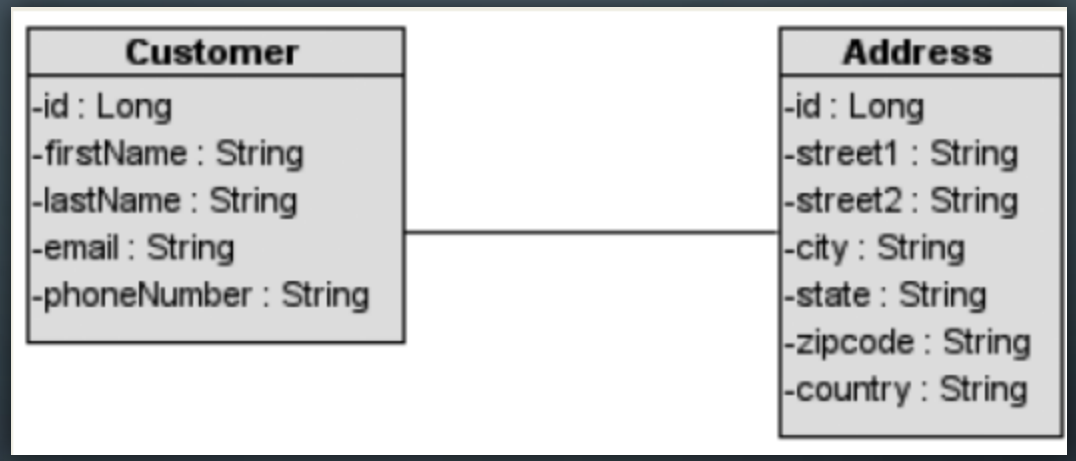
```
@Entity
public class Order {
  @Id @GeneratedValue
  private Long id;
  @Temporal(TemporalType.TIMESTAMP)
  private Date creationDate;
  @OneToMany(fetch = FetchType.EAGER)
  @JoinColumn(name = "order_fk")
  private List<OrderLine> orderLines;
  // Constructors, getters, setters
}
```

TABEL YANG DIHASILKAN @JOINCOLUMN



ASOSIASI BIDIRECTIONAL ANTARA CUSTOMER DAN ADDRESS @MANYTOMANY

- Pada relasi bidirectional, kita perlu mengubah relasi dengan menambahkan atribut Customer ke dalam entitas Address.



- Note: pada diagram class UML, atribut yang merepresentasikan relasi tidak ditunjukkan.

CONTOH KASUS @MANYTOMANY BIDIRECTIONAL

- Contoh: sebuah **Album CD** dapat diciptakan oleh beberapa **Artist,** dan seorang Artist dapat muncul di beberapa Album CD.
- a CD album is created by several artists, and an artist appears on several albums.
- Dengan mengambil asumsi bahwa entitas Artist adalah pemiliki relasi (owner), maka berarti CD berarti bertindak sebagai reverse owner dan harus menggunakan elemen **mappedBy** pada anotasi @ManyToMany.

ONE CD IS CREATED BY SEVERAL ARTISTS

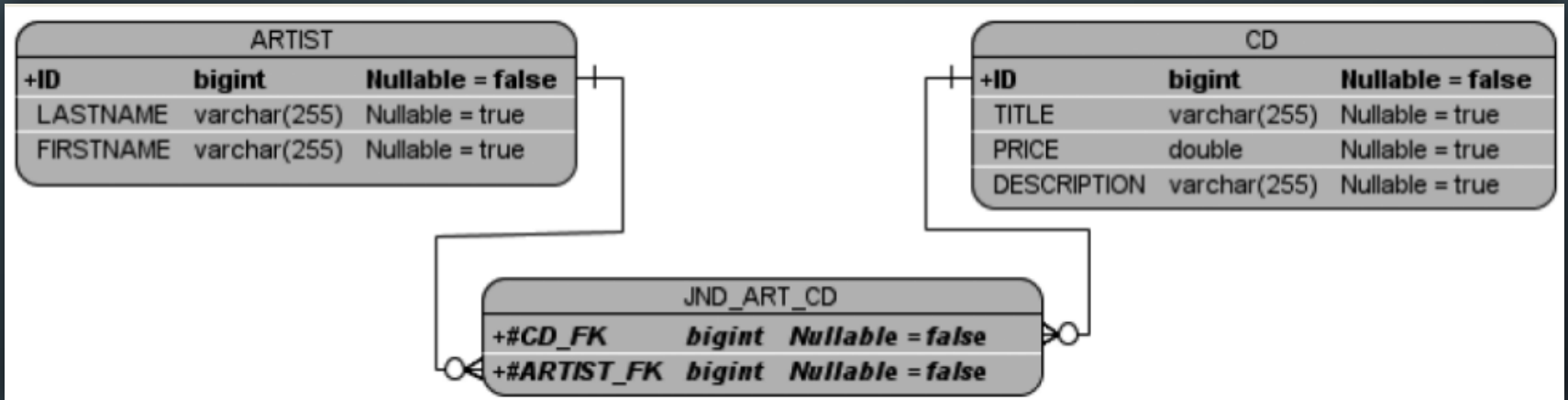
```
@Entity
public class CD {
  @Id @GeneratedValue
  private Long id;
  private String title;
  private Float price;
  private String description;
  @ManyToMany(mappedBy = "appearsOnCDs")
  private List<Artist> createdByArtists;
  // Constructors, getters, setters
}
```

- mappedBy akan memberitahukan engine persistence, bahwa “appearsOnCDs” adalah nama atribut dari entitas pemiliki.

SATU ARTIST DAPAT MUNCUL PADA BANYAK ALBUM CD

```
@Entity
public class Artist {
	@Id @GeneratedValue
	private Long id;
	private String firstName;
	private String lastName;
	@ManyToMany
	@JoinTable(name = "jnd_art_cd",
		joinColumns = @JoinColumn(name = "artist_fk"),
		inverseJoinColumns = @JoinColumn(name = "cd_fk"))
	private List<CD> appearsOnCDs;
	// Constructors, getters, setters
}
```

TABEL YANG DIHASILKAN



FETCHING RELATIONSHIPS

- Parameter “fetch” sangat penting, karena apabila salah dipergunakan, akan menyebabkan masalah performance di aplikasi nanti.
- Setiap anotasi memiliki nilai fetch default yang harus kita waspadai.
- EAGER akan me-load semua data ke dalam memori sehingga pengaksesan database nantinya menjadi minimal.
- LAZY tidak akan memenuhi memori karena kitalah yang menentukan object mana yang perlu di-load. Namun, dengan teknik ini, kita harus mengakses database setiap saat.

Annotation	Default Fetching Strategy
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

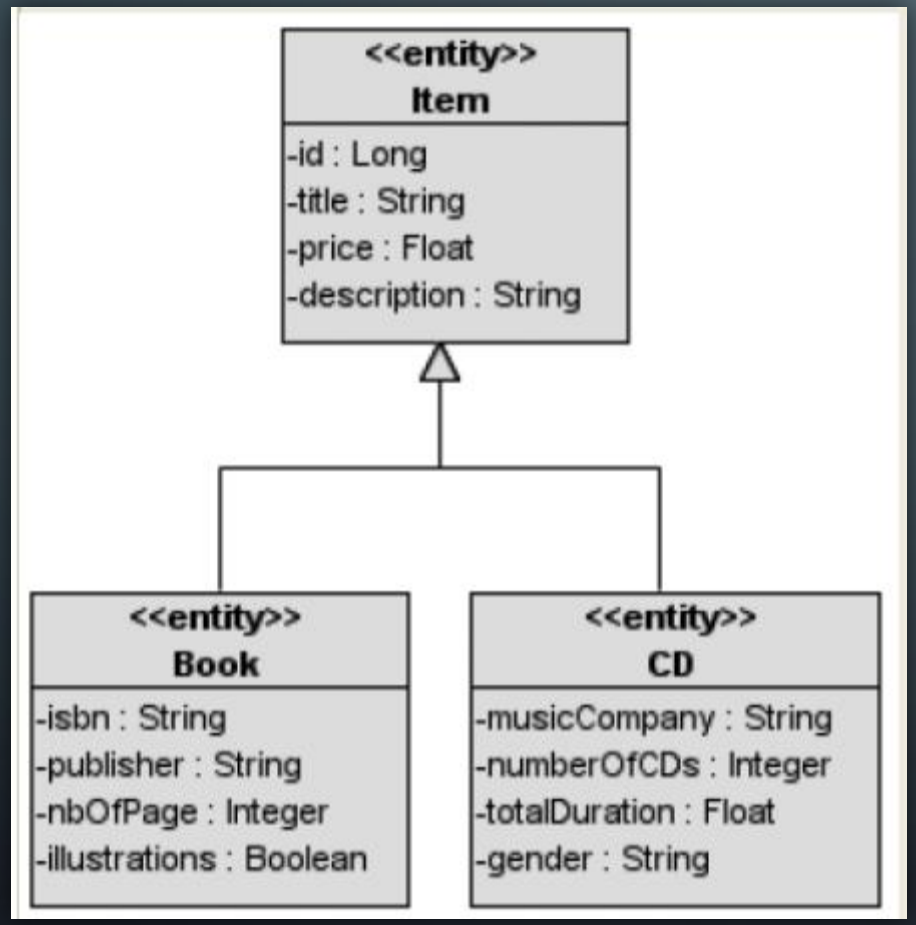
INHERITANCE MAPPING

- Di dunia Java, kita mengenal adanya konsep class interitance/turunan JPA memiliki tida strategi untuk pemetaan inheritance:
  - A **single-table-per-class** hierarchy strategy: semua atribut dari seluruh entitas yang ada dimasukkan menjadi satu tabel (ini merupakan default strategy)
  - A **joined-subclass** strategy: pada cara ini, semua class (abstract maupun concrete) dipetakan menjadi tabel masing-masing.
  - A **table-per-concrete-class** strategy: pada cara ini, setiap entitas concrete dipetakan menjadi tabel sendiri.

CONTOH KASUS INHERITANCE STRATEGIES

- Pada setiap kasus hirarki entitas, pasti ada yang dinamakan entitas ROOT (orang tua).
- Entitas root ini dapat mendefinisikan strategi inheritance dengan menggunakan anotasi @Inheritance.
- Apabila anotasi ini tidak digunakan, maka strategy default-lah yang akan digunakan (*single table per class*)
- Contoh kasus: entitas CD dan Book diturunkan dari entitas Item

ENTITAS CD DAN BOOK, DITURUNKAN DARI ENTITAS ITEM



- Entitas Item adalah entitas root dan memiliki identifier, yang akan menjadi primary key, yang akan diturunkan kepada kedua entitas CD dan Book.

SINGLE-TABLE STRATEGY: ITEM, BOOK, & CD

- Strategi ini adalah default apabila kita tidak menggunakan anotasi @Inheritance, di mana semua entitas yang ada akan dipetakan menjadi SATU TABEL saja.

```
@Entity
public class Item {
  @Id @GeneratedValue
  protected Long id;
  @Column(nullable = false)
  protected String title;
  @Column(nullable = false)
  protected Float price;
  protected String description;
  // Constructors, getters, setters
}
```

```
@Entity
public class Book extends Item {
  private String isbn;
  private String publisher;
  private Integer nbOfPage;
  private Boolean illustrations;
  // Constructors, getters, setters
}
```

```
@Entity
public class CD extends Item {
  private String musicCompany;
  private Integer numberOfCDs;
  private Float totalDuration;
  private String gender;
  // Constructors, getters, setters
}
```

TABEL YANG DIHASILKAN

- Dengan menggunakan strategy single-table, maka semua entitas akan masuk ke dalam satu tabel dengan nama defaultnya adalah nama dari entitas root, yaitu ITEM.

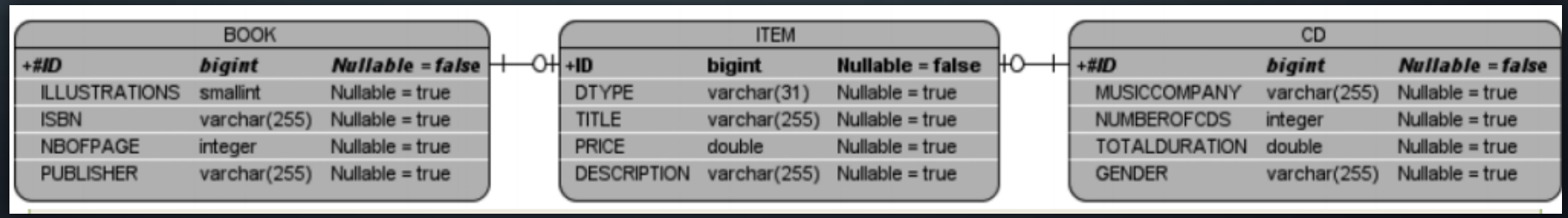
ITEM		
+ID	**bigint**	**Nullable = false**
DTYPE	varchar(31)	Nullable = true
TITLE	varchar(255)	Nullable = false
PRICE	double	Nullable = false
DESCRIPTION	varchar(255)	Nullable = true
ILLUSTRATIONS	smallint	Nullable = true
ISBN	varchar(255)	Nullable = true
NBOFPAGE	integer	Nullable = true
PUBLISHER	varchar(255)	Nullable = true
MUSICCOMPANY	varchar(255)	Nullable = true
NUMBEROFCDS	integer	Nullable = true
TOTALDURATION	double	Nullable = true
GENDER	varchar(255)	Nullable = true

JOINED STRATEGY

- Pada strategy ini, setiap entitas akan dipetakan menjadi tabel sendiri-sendiri.
- Entitas root akan menjadi tabel yang berisikan primary key yang akan digunakan oleh semua tabel turunannya. Selain itu, entitas root ini juga akan mendefinisikan kolom discriminator.
- Semua subclass turunan akan menjadi tabel tersendiri yang berisikan atribut-atribut yang dimilikinya, plus primary key yang mengacu pada primary key entitas root.
- Tabel yang bukan root tidak akan memiliki kolom discriminator.

ENTITAS ITEM MENGGUNAKAN JOINED STRATEGY (CD & BOOK "EXTENDS TO ITEM" TIDAK BERUBAH)

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Item {
  @Id @GeneratedValue
  protected Long id;
  protected String title;
  protected Float price;
  protected String description;
  // Constructors, getters, setters
}
```



BOOK		
+#ID	*bigint*	*Nullable = false*
ILLUSTRATIONS	smallint	Nullable = true
ISBN	varchar(255)	Nullable = true
NBOFPAGE	integer	Nullable = true
PUBLISHER	varchar(255)	Nullable = true

ITEM		
+ID	bigint	Nullable = false
DTYPE	varchar(31)	Nullable = true
TITLE	varchar(255)	Nullable = true
PRICE	double	Nullable = true
DESCRIPTION	varchar(255)	Nullable = true

CD		
+#ID	*bigint*	*Nullable = false*
MUSICCOMPANY	varchar(255)	Nullable = true
NUMBEROFCDS	integer	Nullable = true
TOTALDURATION	double	Nullable = true
GENDER	varchar(255)	Nullable = true

- Note: kolom "DTYPE" adalah kolom discriminator
- Kita dapat melakukan customizing nama discrimator (lihat di buku lebih detil)

TABLE-PER-CLASS STRATEGY

- Pada stragety ini, setiap entitas akan dipetakan menjadi tabel sendiri-sendiri sama seprti pada joined strategy.
- Perbedaanya adalah bahwa semua atribut dari entitas root akan juga dipetakan menjadi kolom-kolom di dalam tabel turunannya.
- Dari segi database, hal ini adalah model yang tidak normal (denormalize).
- Dengan strategy ini, tidak ada tabel yang di-share, tidak ada kolom yang di-share, dan tidak ada kolom discriminator. Yang dibutuhkan untuk relasi hanyalah bahwa semua tabel harus memiliki primary key yang cocok/sama dengan tabel lainnya.

ENTITAS ITEM MENGGUNAKAN TABLE-PER-CLASS STRATEGY (CD & BOOK "EXTENDS TO ITEM" TIDAK BERUBAH)

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Item {
  @Id @GeneratedValue
  protected Long id;
  protected String title;
  protected Float price;
  protected String description;
  // Constructors, getters, setters
}
```

HASIL:
TABEL BOOK DAN CD MEMILIKI KOLOM YG SAMA DENGAN ITEM

- Pada gambar ini, kita lihat bahwa BOOK dan CD menduplikasi kolom ID, TITLE, PRICE, dan DESCRIPTION yang dimiliki oleh tabel ITEM.
- Note: tidak ada hubungan antar tabel

BOOK		
+ID	**bigint**	**Nullable = false**
TITLE	varchar(255)	Nullable = true
PRICE	double	Nullable = true
ILLUSTRATIONS	smallint	Nullable = true
DESCRIPTION	varchar(255)	Nullable = true
ISBN	varchar(255)	Nullable = true
NBOFPAGE	integer	Nullable = true
PUBLISHER	varchar(255)	Nullable = true

ITEM		
+ID	**bigint**	**Nullable = false**
TITLE	varchar(255)	Nullable = true
PRICE	double	Nullable = true
DESCRIPTION	varchar(255)	Nullable = true

CD		
+ID	**bigint**	**Nullable = false**
MUSICCOMPANY	varchar(255)	Nullable = true
NUMBEROFCDS	integer	Nullable = true
TITLE	varchar(255)	Nullable = true
TOTALDURATION	double	Nullable = true
PRICE	double	Nullable = true
DESCRIPTION	varchar(255)	Nullable = true
GENDER	varchar(255)	Nullable = true

THAT'S ALL FOR TODAY!

BY : SENDY FERDIAN SUJADI, S.KOM., M.T., CEH, CEI, MTCNA, MTCRE, MTCINE, MTA