Escuela Java – JPA
Parte 2

TOGETHER. FREE YOUR ENERGIES

Mappings Parte I - I

Marcar una clase como entidad persistente

@Entity

```
@Entity
public class Factura implements Serializable {

   Long id;


   @Id
   public Long getId() { return id; }

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

}
```

Mappings Parte I - II

Definir explícitamente la tabla a que mapea la entidad

@Table

```
@Entity  @Table(name="tbl_factura")
public class Factura implements Serializable {

   ...

}
```

```
@Table(name="tbl_factura",
    uniqueConstraints = {@UniqueConstraint(columnNames={"month", "day"})}

)
```

Mappings Parte I - III

Mapeo de Columnas

@Column

```
@Entity
public class Factura implements Serializable {

...

@Column(updatable = false, name = "nombre", nullable = false, length=50)
public String getName() { ... }
```

Opciones

- name="columnName";
- boolean unique default false;
- boolean nullable() default true;
- boolean insertable() default true;
- boolean updatable() default true;
- String columnDefinition() default "";
- String table() default "";
- int length() default 255;
- int precision() default 0;
- int scale() default 0;

Mappings Parte I - IV

Objetos embebidos (Components)

@Embedded

```
@Entity
public class A implements Serializable {

    @Embedded  B b;


    ...
}
```

Mappings Parte I - V

Objetos embebidos (Components)

@Embeddable

```
@Embeddable
public class B implements Serializable {

    String b1;  C c;
}
```

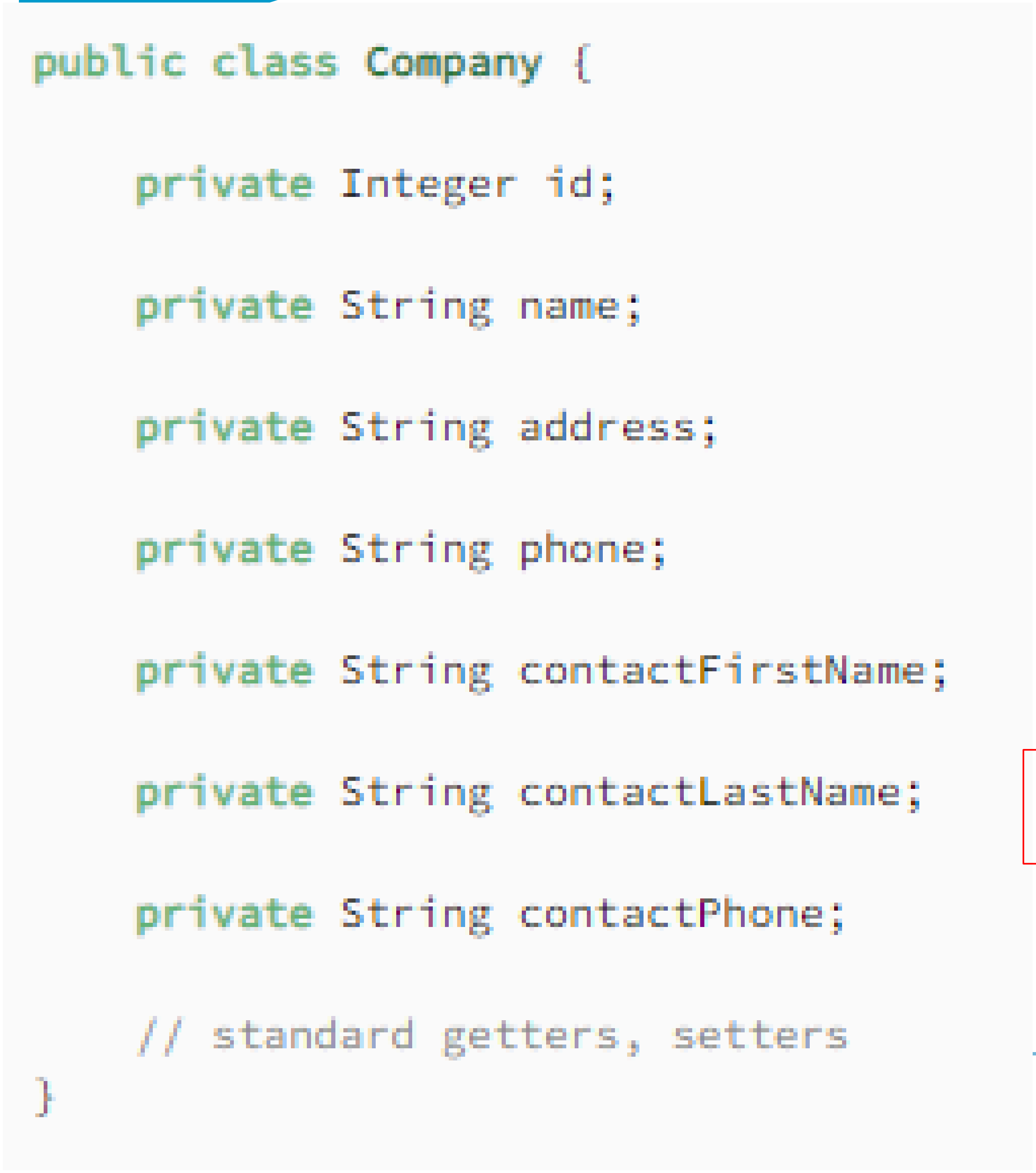
```
@Embeddable
public class C implements Serializable {

    private String c1;

    @Column(name="c2")  private String c2;

    //getters y setters
}
```

Mappings Parte I - V

Objetos embebidos (Components)

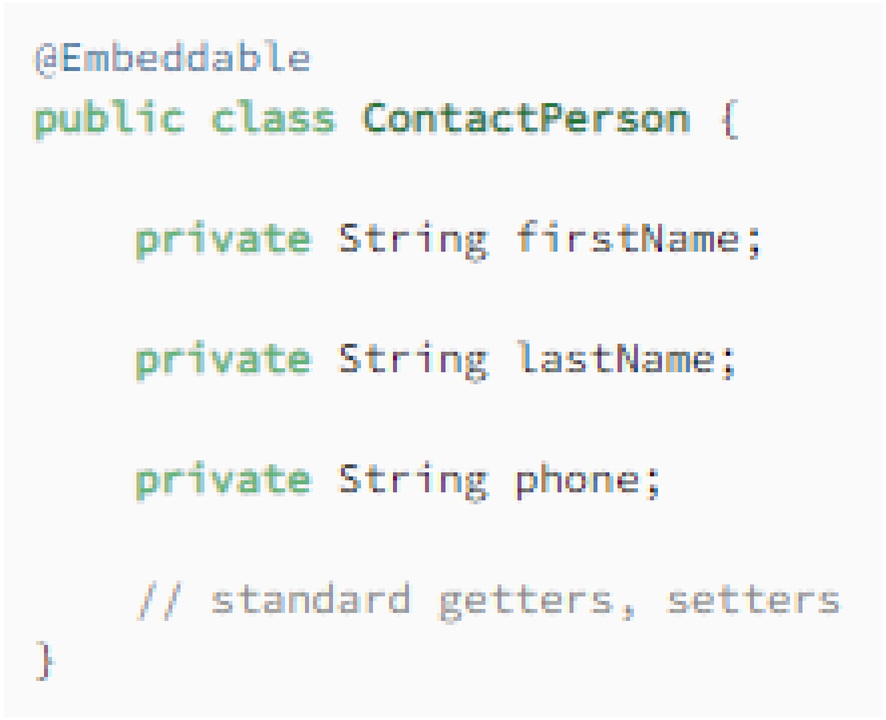
@Embeddable

```
@Embedded  @AttributeOverrides( {
        @AttributeOverride(name="nombre", column = @Column(name="nuevoNombre")

} )
Country bornIn;
```

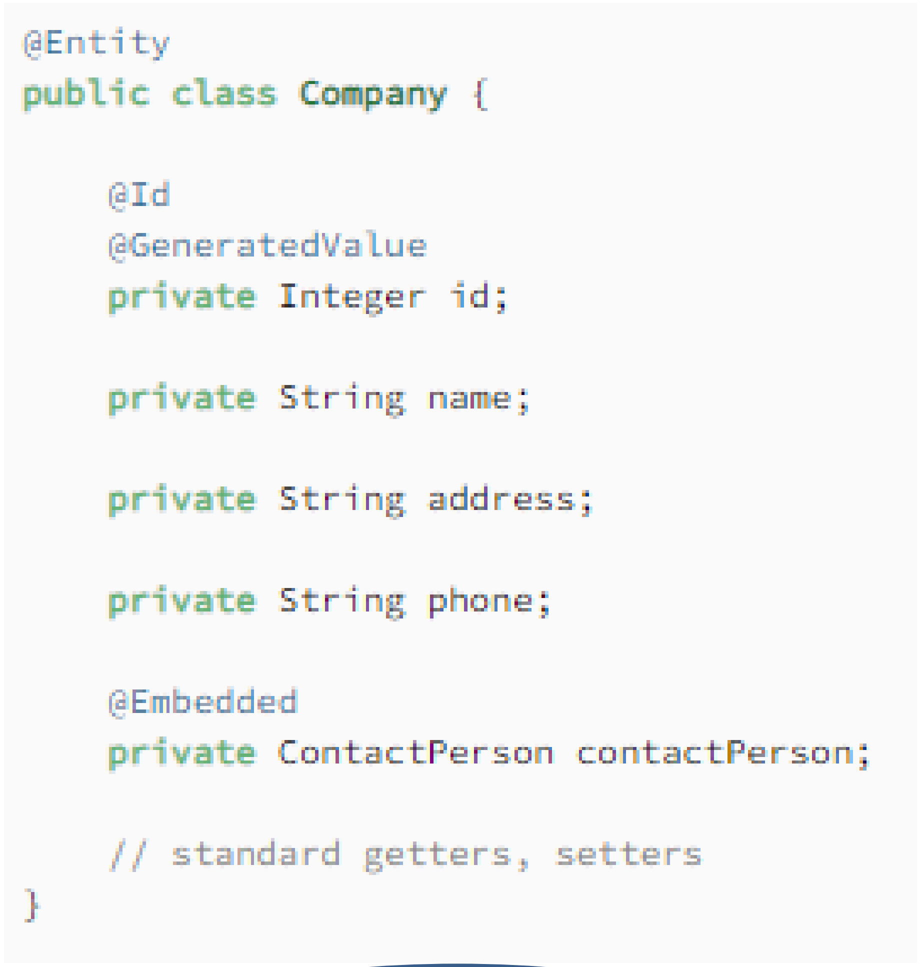


```java
public class Company {

    private Integer id;

    private String name;

    private String address;

    private String phone;

    private String contactFirstName;

    private String contactLastName;

    private String contactPhone;

    // standard getters, setters
}
```

Los datos de la persona de contacto
Deberian ir en una clase separada

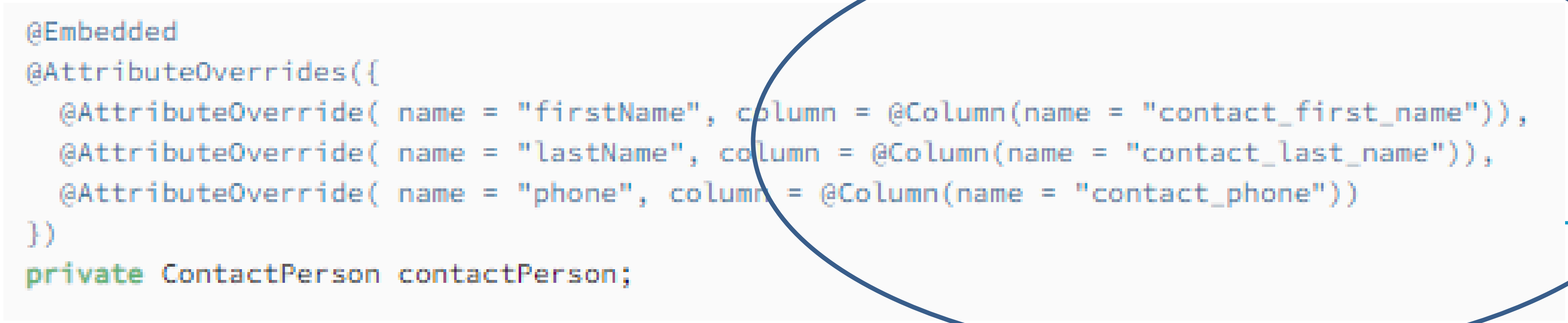


```java
@Embeddable
public class ContactPerson {

    private String firstName;

    private String lastName;

    private String phone;

    // standard getters, setters
}
```



```java
@Entity
public class Company {

    @Id
    @GeneratedValue
    private Integer id;

    private String name;

    private String address;

    private String phone;

    @Embedded
    private ContactPerson contactPerson;

    // standard getters, setters
}
```

Ver la clase original de Company de
La diapositiva anterior



```java
@Embedded
@AttributeOverrides({
  @AttributeOverride( name = "firstName", column = @Column(name = "contact_first_name")),
  @AttributeOverride( name = "lastName", column = @Column(name = "contact_last_name")),
  @AttributeOverride( name = "phone", column = @Column(name = "contact_phone"))
})
private ContactPerson contactPerson;
```

Mappings Parte I - VI

Generación de IDs por parte de Hibernate

Alternativas posibles

- AUTO – Dejá que elija Hibernate qué usar. Identity column, sequence o tabla, dependiendo del motor.
- TABLE – Creá una tabla que tenga el id
- IDENTITY – Usá una Identity column SEQUENCE – Usá
- una Sequence

Mappings Parte I - VII

Generación de IDs por parte de Hibernate

```
@Id
  @GeneratedValue (strategy=GenerationType.AUTO)
  @Column (name="ISBN")
  private long isbn;
```

```
@Entity  @javax.persistence.SequenceGenerator(
    name="SEQ_STORE",
    sequenceName="my_sequence"

)
public class Store implements Serializable {

    private Long id;  @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_STORE")
    public Long getId() { return id; }
}
```

Ejercicio

Ejercicio IV de la guía

Mappings Parte I - VIII

Mapeo de Herencia

**SINGLE_TABLE,
JOINED,
TABLE_PER_CLASS.**

Estrategias posibles

- **Table per Class Hierarchy**
- **Table per Subclass**
- **Table per Concrete Subclass**

Mappings Parte I - IX

El problema

```
public abstract class Disc implements Serializable {
   private static final long serialVersionUID = -5119119376751110049L;
   private Long discId;
   private String name;  private Integer price;
   //getters and setters
}
```

```
public class AudioDisc extends Disc implements Serializable {
private static final long serialVersionUID = -8314602929677976050L;
private Integer noOfSongs;
private String singer;
//getters and setters
}
```

```
public class VideoDisc extends Disc implements Serializable {
private static final long serialVersionUID = -6857479057343664829L;
private String director;
private String language;
//getters and setters
}
```

Mappings Parte I - X

El mapeo

Table per Class Hierarchy

```
@Entity
@Inheritance (strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISC_TYPE",
    discriminatorType=DiscriminatorType.STRING)
public class Disc implements Serializable {

   @Id
   @GeneratedValue (strategy=GenerationType.AUTO)  @Column
   (name="DISC_ID")
   private Long discId;

   @Column (name="NAME")  private String name;

   @Column (name="PRICE")  private Integer price;

   //Getters and Setters
}
```

Mappings Parte I - XI

El mapeo

Table per Class Hierarchy

```
@Entity
@DiscriminatorValue ("AUDIO")
public class AudioDisc extends Disc implements Serializable {

   private static final long serialVersionUID = 8510682776718466795L;

   @Column (name="NO_OF_SONGS")
   private Integer noOfSongs;

   @Column (name="SINGER")  private String singer;
   //Getters and Setters
  }
```

Mappings Parte I - XII

El mapeo

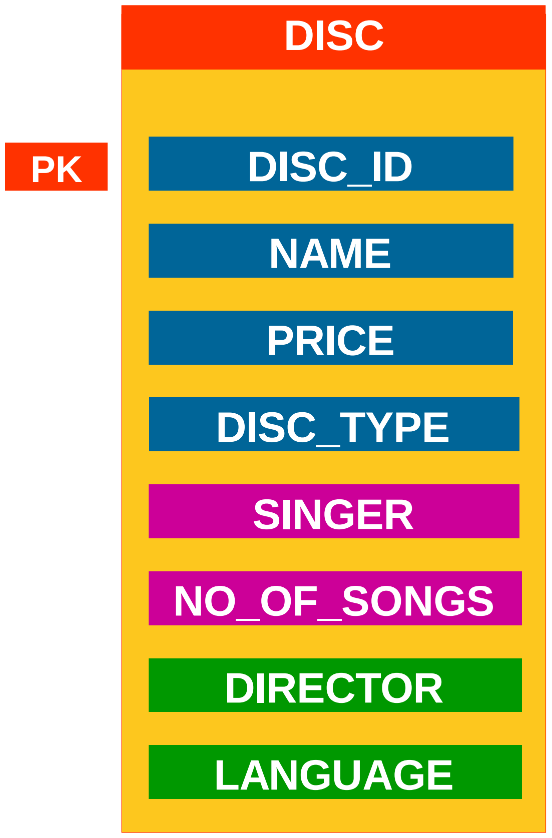
Table per Class Hierarchy

```
@Entity
@DiscriminatorValue ("VIDEO")
public class VideoDisc extends Disc implements Serializable {

  private static final long serialVersionUID =-3637473456207740684L;

  @Column (name="DIRECTOR")
  private String director;  @Column (name="LANGUAGE")  private
  String language;

  //Getters and Setters
 }
```

Mappings Parte I - XIII

El resultado en la BD

Table per Class Hierarchy



Mappings Parte I - XIV

El mapeo

Table per Subclass

```
@Entity
@Inheritance (strategy=InheritanceType.JOINED)
public abstract class Disc implements Serializable {

   private static final long serialVersionUID = 3087285416805917315L;
   @Id
   @GeneratedValue (strategy=GenerationType.AUTO)
   @Column (name="DISC_ID")
   private Long discId;
   @Column (name="NAME")
   private String name;
   @Column (name="PRICE")
   private Integer price;

   //getters and settes
}
```

Mappings Parte I - XV

El mapeo

Table per Subclass

```
@Entity
@Table (name="AUDIO_DISC")
 @PrimaryKeyJoinColumn (name="DISC_ID")
public class AudioDisc extends Disc implements Serializable {

   private static final long serialVersionUID = 8510682776718466795L;

   @Column (name="NO_OF_SONGS")
   private Integer noOfSongs;
   @Column (name="SINGER")  private String singer;
         // getter and setters
  }
```

Mappings Parte I - XVI

El mapeo

Table per Subclass

```
@Entity
@Table (name="VIDEO_DISC")  @PrimaryKeyJoinColumn
(name="DISC_ID")
public class VideoDisc extends Disc implements Serializable {

  private static final long serialVersionUID = -3637473456207740684L;

  @Column (name="DIRECTOR")
  private String director;  @Column (name="LANGUAGE")  private
  String language;

        // getter and setters
}
```

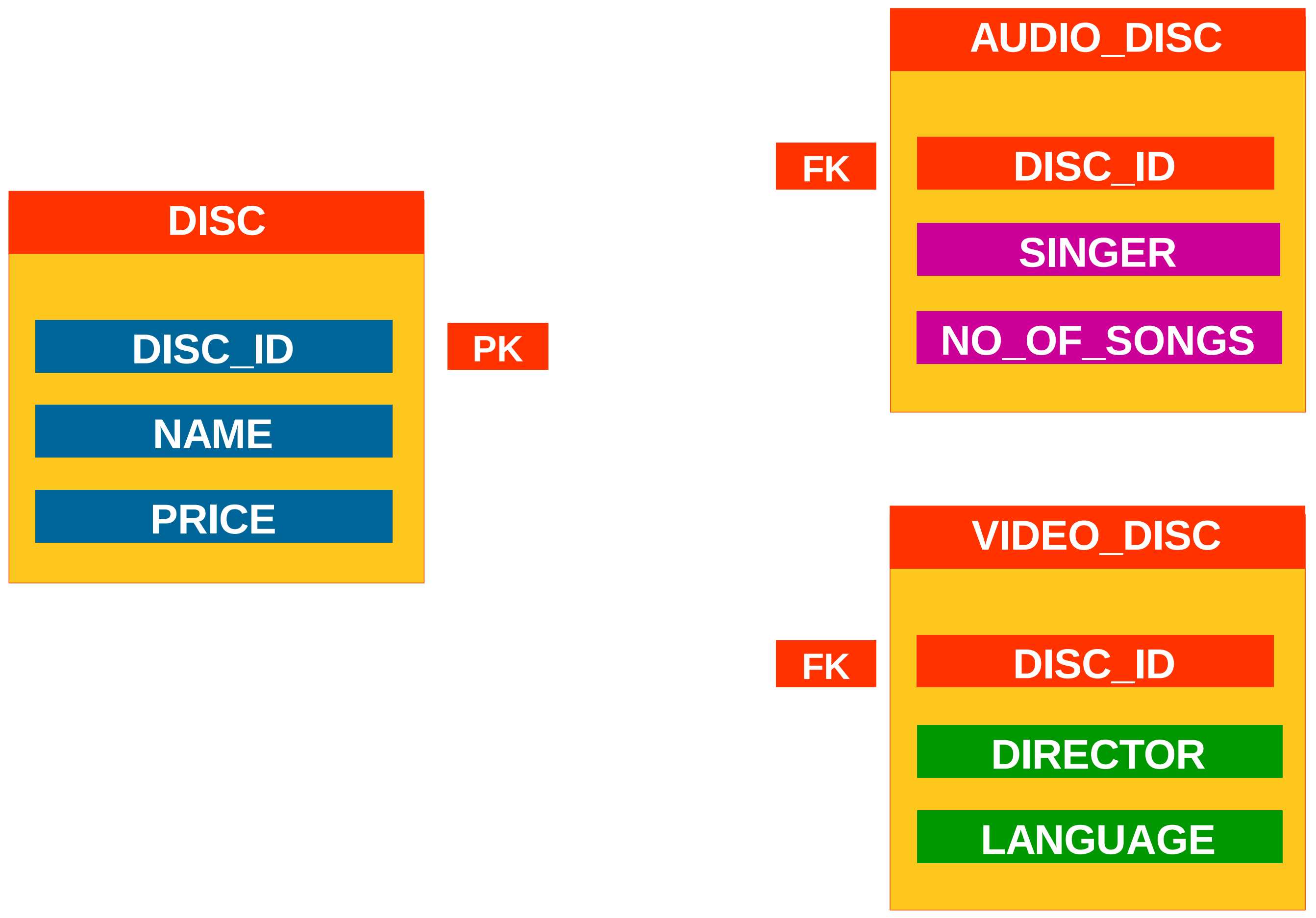
CONSULTING.TECHNOLOGY.OUTSOURCING

Together. Free your energies

Mappings Parte I - XVII

El resultado en la BD

Table per Subclass



Mappings Parte I - XVIII

El mapeo

Table per Concrete Class

```
@Entity
@Inheritance (strategy=InheritanceType.TABLE_PER_CLASS)  public
abstract class Disc implements Serializable {

   private static final long serialVersionUID = 3087285416805917315L;

   @Id
   @GeneratedValue (strategy=GenerationType.AUTO)
   @Column (name="DISC_ID")  private Long discId;  @Column
   (name="NAME")  private String name;  @Column (name="PRICE")
   private Integer price;
   //getters and setters
 }
```

Mappings Parte I - XIX

El mapeo

Table per Concrete Class

```
@Entity
@Table (name="AUDIO_DISC")
public class AudioDisc extends Disc implements Serializable {

   private static final long serialVersionUID = 8510682776718466795L;

   @Column (name="NO_OF_SONGS")
   private Integer noOfSongs;  @Column (name="SINGER")  private
   String singer;
   //getters and setters
}
```

Mappings Parte I - XX

El mapeo

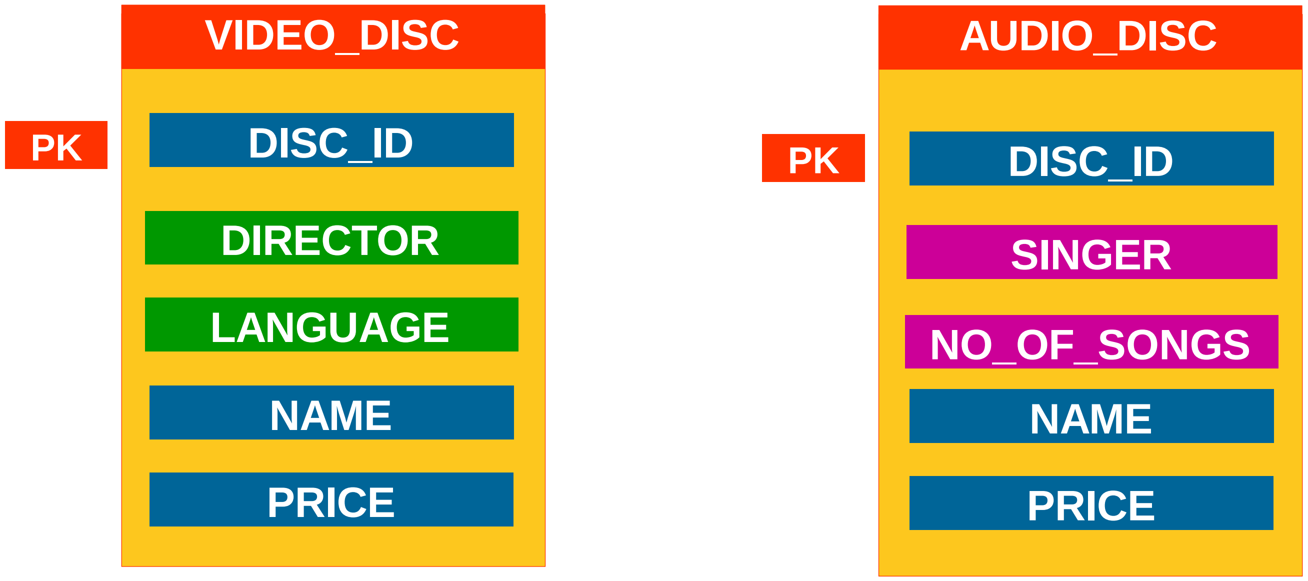
Table per Concrete Class

```
@Entity
@Table (name="VIDEO_DISC")
 public class VideoDisc extends Disc implements Serializable {

  private static final long serialVersionUID = -3637473456207740684L;

  @Column (name="DIRECTOR")
  private String director;  @Column (name="LANGUAGE")  private
  String language;

         // getter and setters
}
```

Mappings Parte I - XXI

El resultado en la BD

Table per Concrete Class



Ejercicio

Ejercicio V de la guía

Muchas gracias!