

Hibernate JPA

JPA

Java Persistence API



HIBERNATE



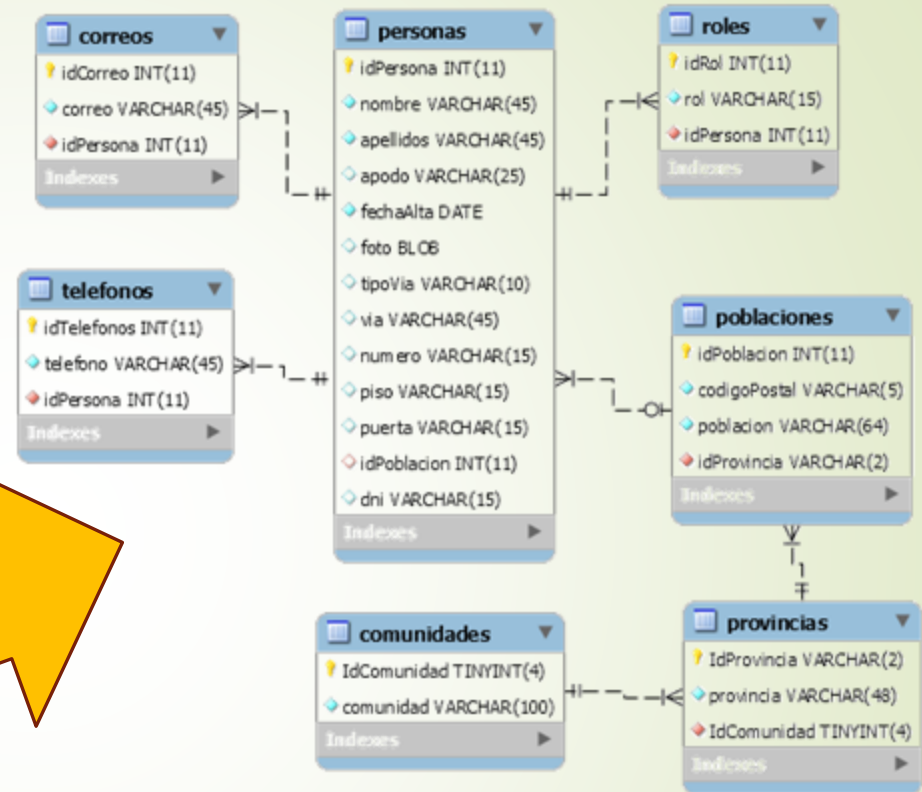
Presentaciones

- Java ?
 - O.O. ?
 - Interfaces / Polimorfismo ?
 - Acoplamiento - Dependencias ?
- JDBC ?
- Patrones de Diseño ?
 - Singleton ?
 - Value Object ?
 - DAO ?
- Frameworks ?

ORM

```
<<Java Class>>
Persona
sysRugby.modelo

serialVersionUID: long
idPersona: int
apellidos: String
apodo: String
fechaAlta: Date
foto: Blob
nombre: String
numero: String
piso: String
puerta: String
tipoVia: String
via: String
dni: String
correos: List<String>
telefonos: List<String>
poblacion: Poblacion
roles: List<Rol>
```



ORM

Object Relational Mapping



1999	EJB 1.0
2001	EJB 2.0
2002	JDO
2002	Hibernate
2006	JPA 1.0 (EJB 3.0) – JAVA EE5
2009	JPA 2.0 – JAVA EE6
2013	JPA 2.1 – JAVA EE7
2017	JPA 2.2
2020	JPA 3.0

HIBERNATE / JPA

JPA es un estándar de Sun (Oracle) que define un API para los ORM en Java. Es una especificación.

Los entity beans de los servidores Java EE se declaran siguiendo el api JPA.

HIBERNATE es un ORM de RedHat, creado por Gavin King, luego JBoss contrató al grupo de desarrollo y dio soporte al proyecto.

Implementa el estándar JPA

IMPLEMENTACIONES JPA	
Hibernate	RedHat
Eclipselink	Eclipse Foundation
Toplink	Oracle
ObjectDB	
CocoBase	
OpenJPA	Apache
Kodo	Bea

CARACTERÍSTICAS

- Las Entity class deben cumplir el estándar JavaBean
 - Son clases simples, no necesitan heredar de otra clase, sólo implementa Serializable
 - Constructor por defecto (explícito o implícito)
 - Debe ser de primer nivel (no interna)
 - No ser final
 - Implementa Serializable
 - Getter y Setter para sus atributos
- Configuración
 - Archivo de Configuración XML (HIBERNATE o JPA)
- Mapeo (Metadatos):
 - Anotaciones (Utiliza el estándar JPA)
 - Archivo de Mapeo XML (HIBERNATE o JPA)

Configuración

Mapeo

HIBERNATE

JPA

<xml>

CONFIG
Hibernate

<xml>

CONFIG
JPA
persistence.xml

<xml>

MAPEO
Hibernate

Java
@JPA

MAPEO

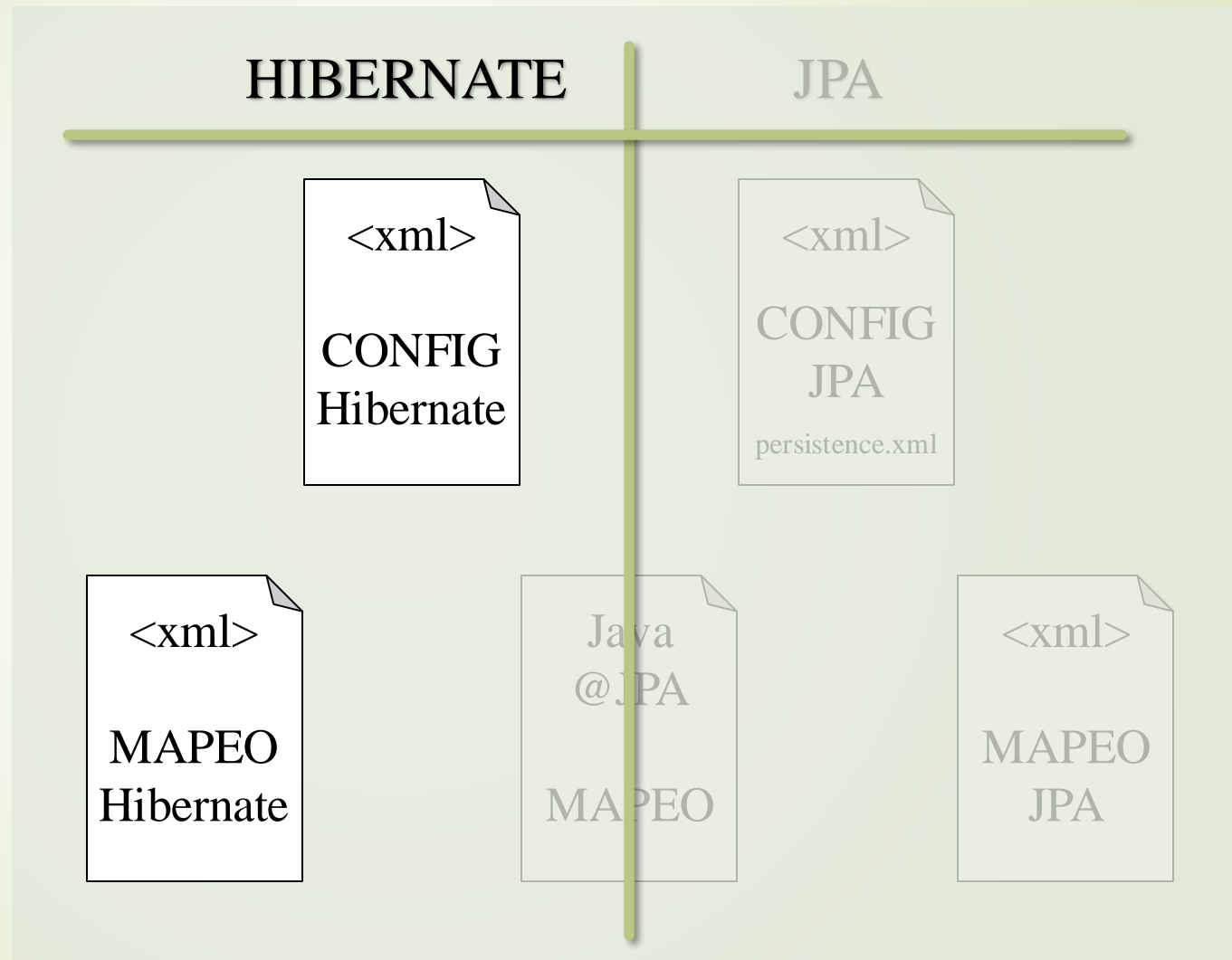
<xml>

MAPEO
JPA

HIBERNATE – Mapeo xml

Configuración

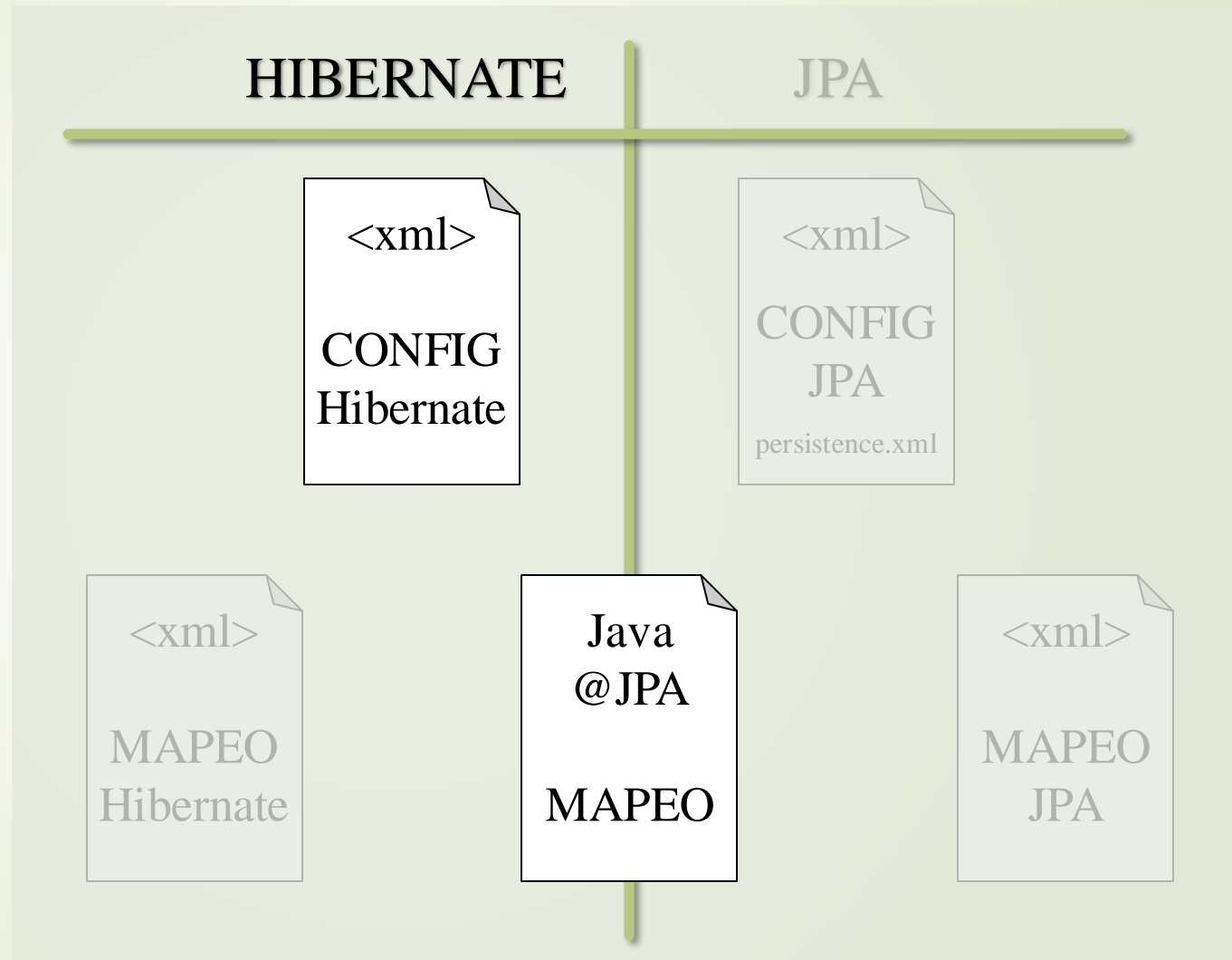
Mapeo



HIBERNATE – Anotaciones JPA

Configuración

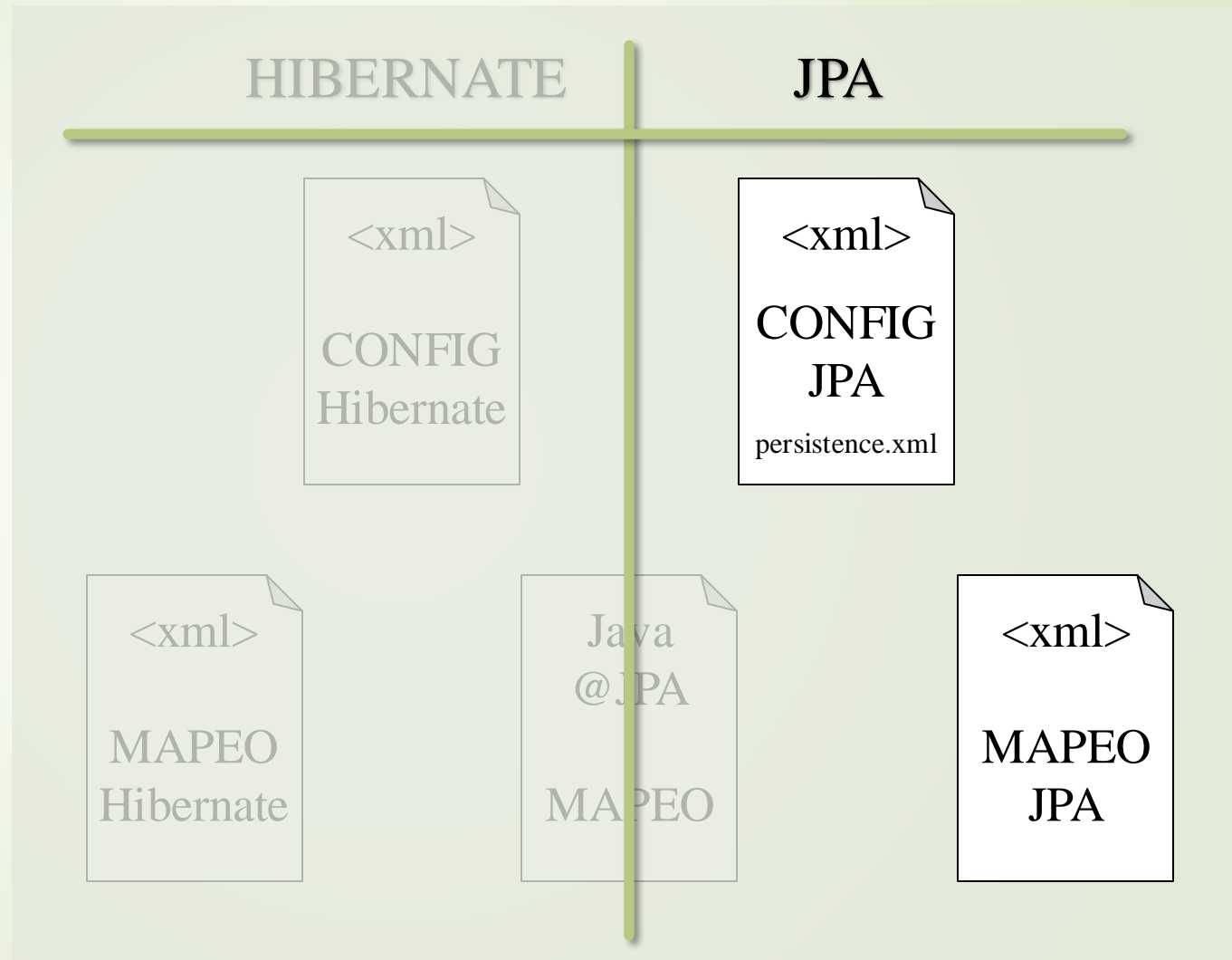
Mapeo



JPA – Mapeo xml

Configuración

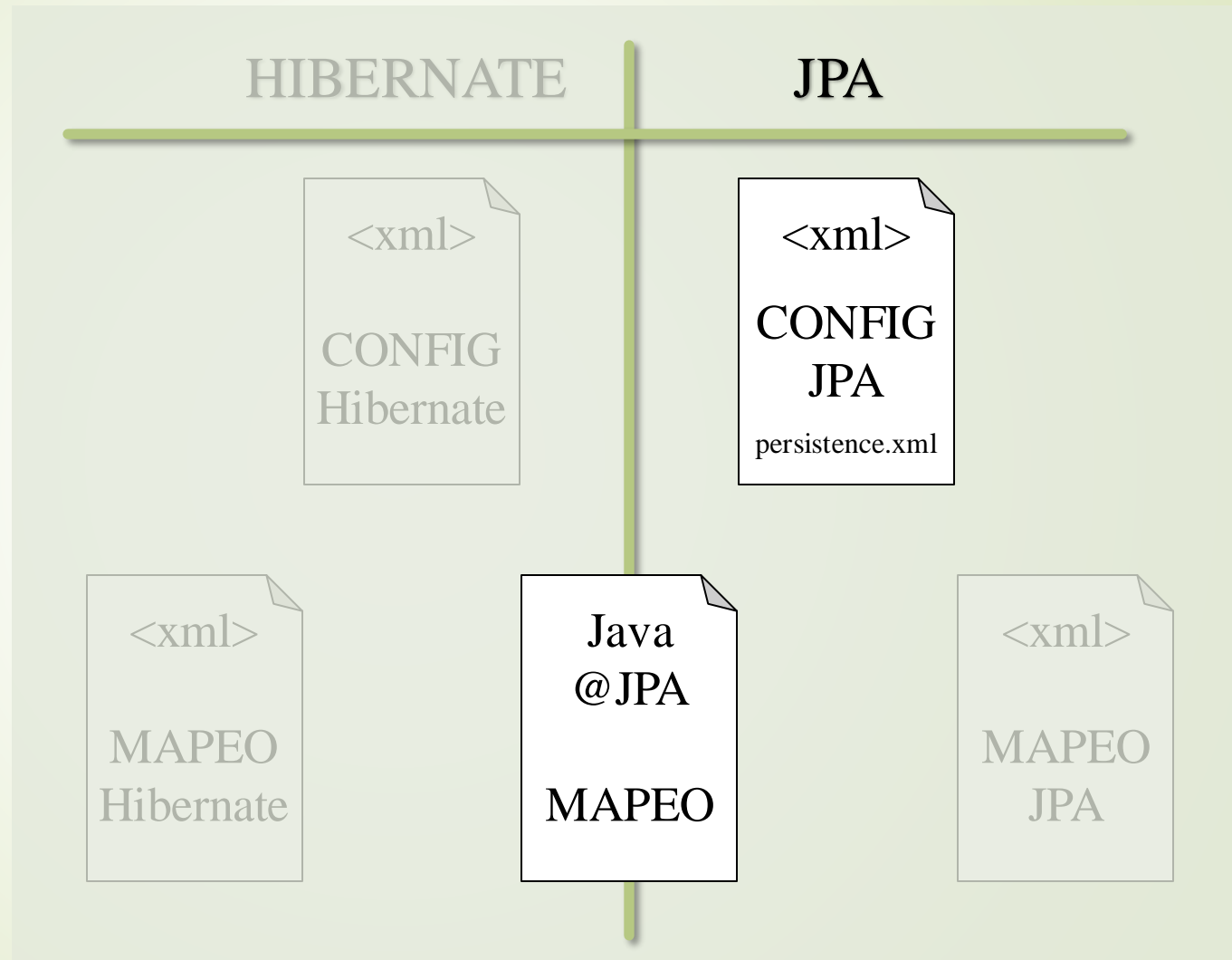
Mapeo



JPA – Anotaciones JPA

Configuración

Mapeo



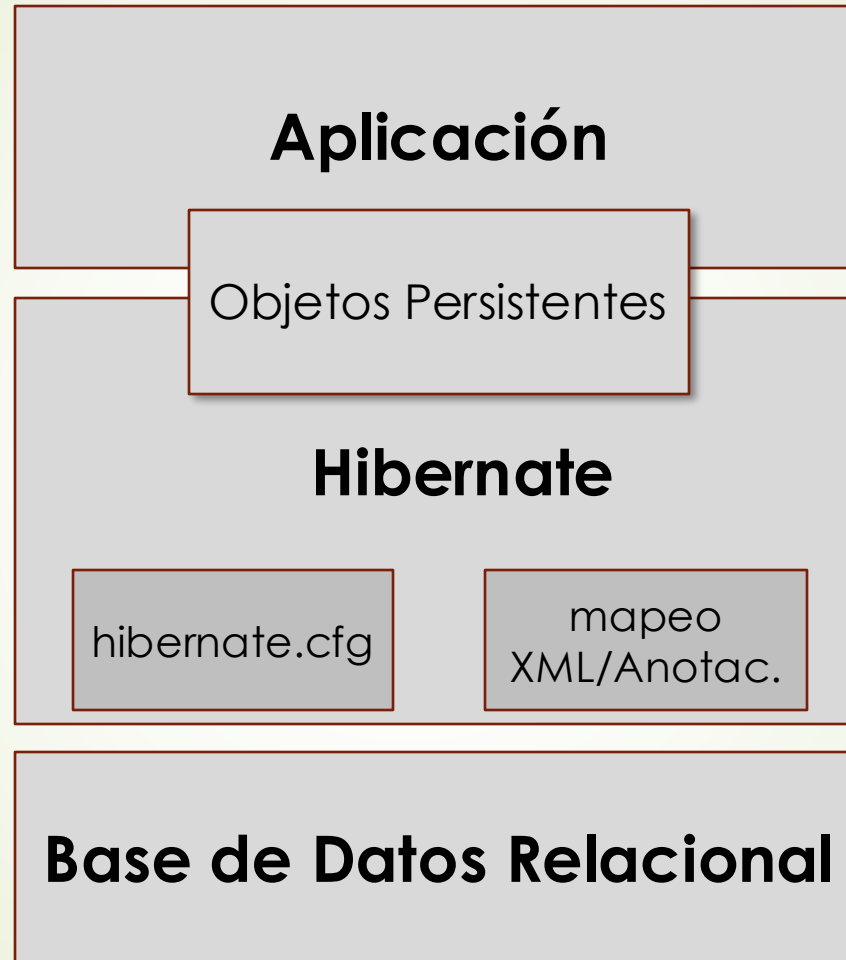
Mapeo

ANOTACIONES	
+	Todo en el mismo lugar
-	Es necesario recompilar
-	Menos portable (si no se quiere utilizar JPA)

DESCRIPTOR XML	
-	Mantenimiento de ficheros externos (1 x clase)
+	No es necesario recompilar

Si existen ambos, prevalece el Descriptor XML

Hibernate





Hibernate 4

Bibliotecas

`\hibernate-release-4.3.7.Final\lib\required*`

`antlr-2.7.7.jar`

`archs`

`dom4j-1.6.1.jar`

`hibernate-commons-annotations-4.0.5.Final.jar`

`hibernate-core-4.3.7.Final.jar`

`hibernate-jpa-2.1-api-1.0.0.Final.jar`

`jandex-1.1.0.Final.jar`

`javassist-3.18.1-GA.jar`

`jboss-logging-3.1.3.GA.jar`

`jboss-logging-annotations-1.2.0.Beta1.jar`

`jboss-transaction-api_1.2_spec-1.0.0.Final.jar`

`\hibernate-release-4.3.7.Final\lib\jpa*`

`hibernate-entitymanager-4.3.7.Final.jar`

HIBERNATE – hibernate.cfg.xml

En la aplicación se debe incluir el archivo hibernate.cfg.xml donde se configura Hibernate. Ej. mapeo con ficheros xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

    <session-factory>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <property name="hibernate.show_sql">true</property>

        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/curso_hibernate</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>

        <mapping resource="persist-xml/introduccion/persona.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```


HIBERNATE – hibernate.cfg.xml

En la aplicación se debe incluir el archivo hibernate.cfg.xml donde se configura Hibernate. Ej. mapeo con anotaciones

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

    <session-factory>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <property name="hibernate.show_sql">true</property>

        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/curso_hibernate</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>

        <mapping class="introduccion.Persona"/>
    </session-factory>

</hibernate-configuration>
```

JPA - persistence.xml

En la carpeta **META-INF** se debe incluir el archivo persistence.xml con la configuración de JPA. Ej. mapeo con ficheros xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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_2_0.xsd" version="2.0">

  <persistence-unit name="UnidadAgenda" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <mapping-file>persist-xml/introduccion/Persona.xml</mapping-file>

    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />
      <property name="hibernate.connection.pool_size" value="1" />

      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
    </properties>
  </persistence-unit>

</persistence>
```

JPA - persistence.xml

En la carpeta **META-INF** se debe incluir el archivo persistence.xml con la configuración de JPA. Ej. mapeo con anotaciones

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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_2_0.xsd" version="2.0">

  <persistence-unit name="UnidadAgenda" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <class>introduccion.Persona</class> <!-- OPCIONAL-->

    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />
      <property name="hibernate.connection.pool_size" value="1" />

      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.url" value="jdbc:mysql://localhost/curso_hibernate"/>
      <property name="hibernate.connection.username" value="root"/>
      <property name="hibernate.connection.password" value="root"/>
    </properties>
  </persistence-unit>

</persistence>
```

HIBERNATE – Mapeo xml

<hibernate-mapping> <class> <id> <property>

Ej. persona.hbm.xml

```
<<Java Class:
G Person:
(default package)

idPersona: int
apellidos: String
apodo: String
nombre: String
dni: String

Persona()
getIdPersona():int
setIdPersona(int)
getApellidos():String
setApellidos(String)
getApodo():String
setApodo(String)
getNombre():String
setNombre(String)
getDni():String
setDni(String):void
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//
Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-
mapping-3.0.dtd">
```

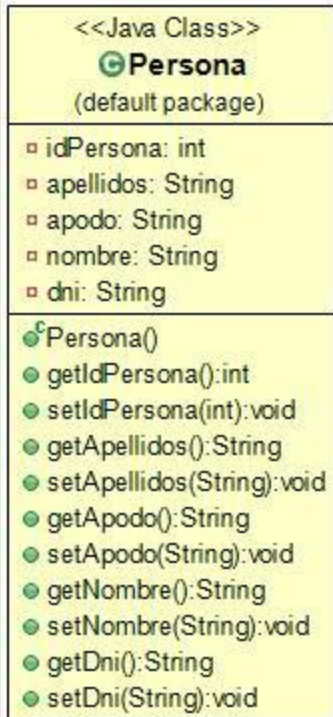
```
<hibernate-mapping>
  <class name="introduccion.Persona">
    <id name="idPersona" type="integer" />
    <property name="apellidos" />
    <property name="apodo" />
    <property name="nombre" />
    <property name="dni" />
  </class>
</hibernate-mapping>
```

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

HIBERNATE – Mapeo xml

<hibernate-mapping> <class> <id> <property>

Ej. persona.hbm.xml



Java Class: **Persona** (default package)

- idPersona: int
- apellidos: String
- apodo: String
- nombre: String
- dni: String

Methods:

- Persona()
- getIdPersona():int
- setIdPersona(int):void
- getApellidos():String
- setApellidos(String):void
- getApodo():String
- setApodo(String):void
- getNombre():String
- setNombre(String):void
- getDni():String
- setDni(String):void

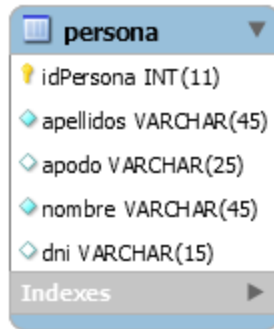


Table: **persona**

- idPersona INT(11)
- apellidos VARCHAR(45)
- apodo VARCHAR(25)
- nombre VARCHAR(45)
- dni VARCHAR(15)

Indexes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-
//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-
mapping-3.0.dtd">
```

```
<hibernate-mapping>
  <class name="introduccion.Persona">
    <id name="idPersona" type="integer" />
    <property name="apellidos" />
    <property name="apodo" />
    <property name="nombre" />
    <property name="dni" />
  </class>
</hibernate-mapping>
```

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

JPA - @Entity / @Id

@Entity Refiere a la clase e indica que será una entidad
@Id Indica el atributo que se corresponde con la PK

UML Class Diagram	Java Code
<pre><<Java Class>> classDiagram class Persona { idPersona int apellidos String apodo String nombre String dni String } Persona() getIdPersona() int setIdPersona(int) getApellidos() String setApellidos(String) getApodo() String setApodo(String) getNombre() String setNombre(String) getDni() String setDni(String) void</pre>	<pre>import java.io.Serializable; import javax.persistence.*; @Entity public class Persona implements Serializable { @Id private int idPersona; private String apellidos; private String apodo; private String nombre; private String dni; public Persona() { } public int getIdPersona() { return idPersona; } ...</pre>

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

JPA - @Entity / @Id

@Entity Refiere a la clase e indica que será una entidad
@Id Indica el atributo que se corresponde con la PK

<<Java Class>>	
Persona (default package)	
idPersona: int	
apellidos: String	
apodo: String	
nombre: String	
dni: String	
Persona() getIdPersona():int setIdPersona(int):void getApellidos():String setApellidos(String):void getApodo():String setApodo(String):void getNombre():String setNombre(String):void getDni():String setDni(String):void	

persona	
idPersona	INT(11)
apellidos	VARCHAR(45)
apodo	VARCHAR(25)
nombre	VARCHAR(45)
dni	VARCHAR(15)
Indexes	

```
import java.io.Serializable;
import javax.persistence.*;

@Entity
public class Persona implements Serializable {
    @Id
    private int idPersona;
    private String apellidos;
    private String apodo;
    private String nombre;
    private String dni;

    public Persona() {
    }

    public int getIdPersona() {
        return idPersona;
    }

    ...
}
```

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

JPA – Mapeo xml

<entity> <attributes> <id>

Ej. Persona.xml

<pre><<Java Class: G Persona (default package) idPersona: int apellidos: String apodo: String nombre: String dni: String Persona() getIdPersona():int setIdPersona(int) getApellidos():String setApellidos(String) getApodo():String setApodo(String) getNombre():String setNombre(String) getDni():String setDni(String):void</pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <entity-mappings xmlns="http://java.sun.org/xml/ns/jpa" version="2.0"> <description> Mapeo minimo de una Entidad persistente </description> <entity class="jpa.Persona"> <attributes> <id name="idPersona" /> </attributes> </entity> </entity-mappings></pre>
--	--

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

JPA – Mapeo xml

<entity> <attributes> <id>

Ej. Persona.xml

<<Java Class>>	
Persona (default package)	
idPersona: int	
apellidos: String	
apodo: String	
nombre: String	
dni: String	
Persona()	
getIdPersona():int	
setIdPersona(int):void	
getApellidos():String	
setApellidos(String):void	
getApodo():String	
setApodo(String):void	
getNombre():String	
setNombre(String):void	
getDni():String	
setDni(String):void	

persona	
idPersona INT(11)	
apellidos VARCHAR(45)	
apodo VARCHAR(25)	
nombre VARCHAR(45)	
dni VARCHAR(15)	
Indexes	

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.org/xml/ns/jpa"
version="2.0">
```

```
<description>
```

Mapeo minimo de una Entidad persistente

```
</description>
```

```
<entity class="jpa.Persona">
```

```
<attributes>
```

```
<id name="idPersona" />
```

```
</attributes>
```

```
</entity>
```

```
</entity-mappings>
```

Todas las propiedades serán mapeadas a una tabla con el mismo nombre de la clase y cada propiedad a una columna homónima

HIBERNATE - Session

```
SessionFactory sessionFactory;  
Session session;  
Configuration configuration = new Configuration();  
configuration.configure("persist-xml/hibernate.cfg.xml");  
  
ServiceRegistry serviceRegistry;  
serviceRegistry = new StandardServiceRegistryBuilder()...  
  
sessionFactory =  
configuration.buildSessionFactory(serviceRegistry);  
  
session=sessionFactory.openSession();  
...  
session.close();  
sessionFactory.close();
```

- **SessionFactory**
- **Session**
- **ServiceRegistry**
 - Se obtiene de `StandardServiceRegistryBuilder()`
- **Configuration**
 - `.configure(String xml)`
 - `SessionFactory .buildSessionFactory(ServiceRegistry sr)`
- **SessionFactory**
 - `Session .openSession()`

HIBERNATE - Session

- id **save**(Object o)
- void **update**(Object o)
- void **saveOrUpdate**(Object o)
- void **delete**(Object o)
- Object **get**(Class c , Object pk)
- Crear Queries (vs. métodos)
- void **flush**()
- Connection **close**()

```
session=sessionFactory.openSession();

Persona p = new Persona();
p.setApellidos("...");

session.beginTransaction();
session.save(p);
session.getTransaction().commit();

Persona otra = (Persona)session.get(Persona.class, 27);
System.out.println(otra.getApodo());

session.close();
sessionFactory.close();
```

JPA - Entity Manager

```
EntityManagerFactory emf;  
EntityManager em;  
  
emf =  
Persistence.createEntityManagerFactory("ejemplos_curso");  
em = emf.createEntityManager();  
  
em.close();  
emf.close();
```

- **EntityManagerFactory**
- **EntityManager**
- **Persistence**
 - *.createEntityManagerFactory(persistent_unit) EntityManagerFactory*
- **EntityManagerFactory**
 - *.createEntityManager() EntityManager*

JPA - EntityManager

- void **persist**(Object o)
- <T> **merge**(<T> o)
- void **remove**(Object o)
- <T> **find**(Class<T> , Object pk)
- EntityManager **getTransaction**()
- Crear Queries (vs. métodos)
- void **flush**()
- void **close**()

```
em = emf.createEntityManager();
EntityManager tx = em.getTransaction();

Persona p = new Persona();
p.setApellidos("...");

tx.begin();
em.persist(p);
tx.commit();

Persona otra = em.find(Persona.class, 27);
System.out.println(otra.getApodo());

em.close();
emf.close();
```

Contexto de Persistencia

Estado de las Entidades



new



managed



detached



removed



No esta asociado a ningún PersistenceContext



Asociado a un PersistenceContext



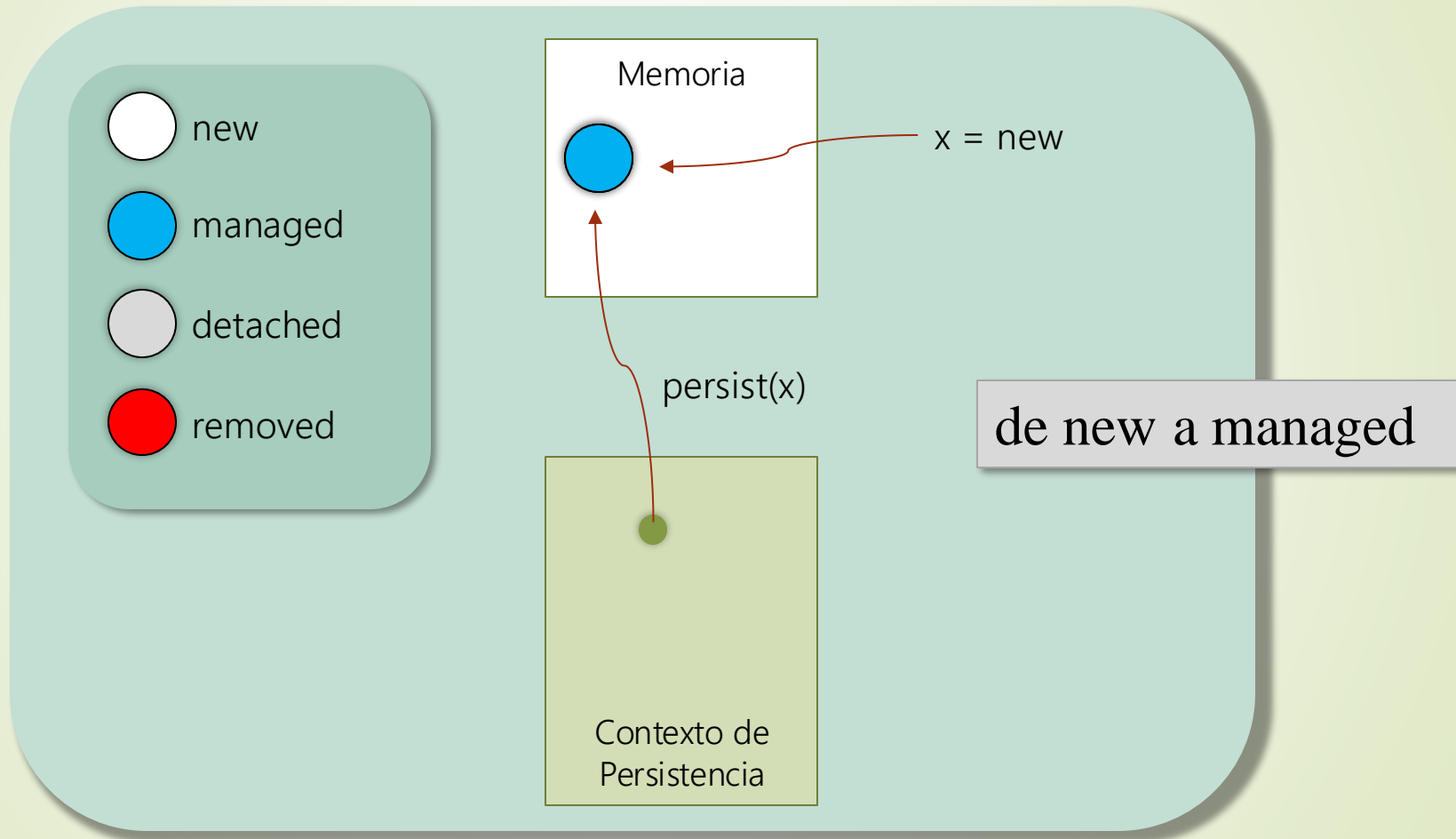
Ha estado asociado al PersistenceContext y deja de estarlo



Controlado por el PersistenceContext pero va a ser eliminado de la base de datos

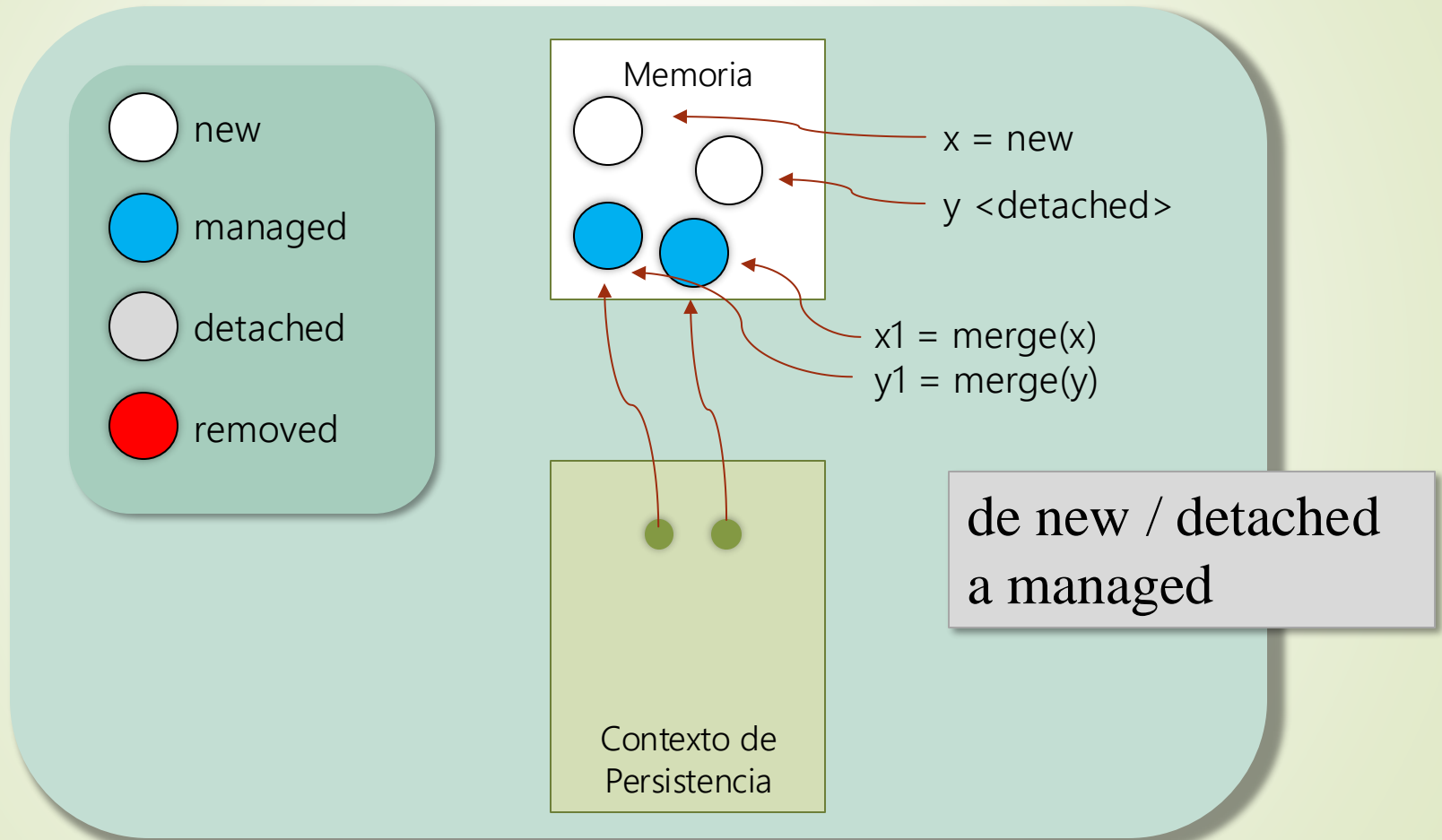
persist()

Convierte una nueva instancias en persistente



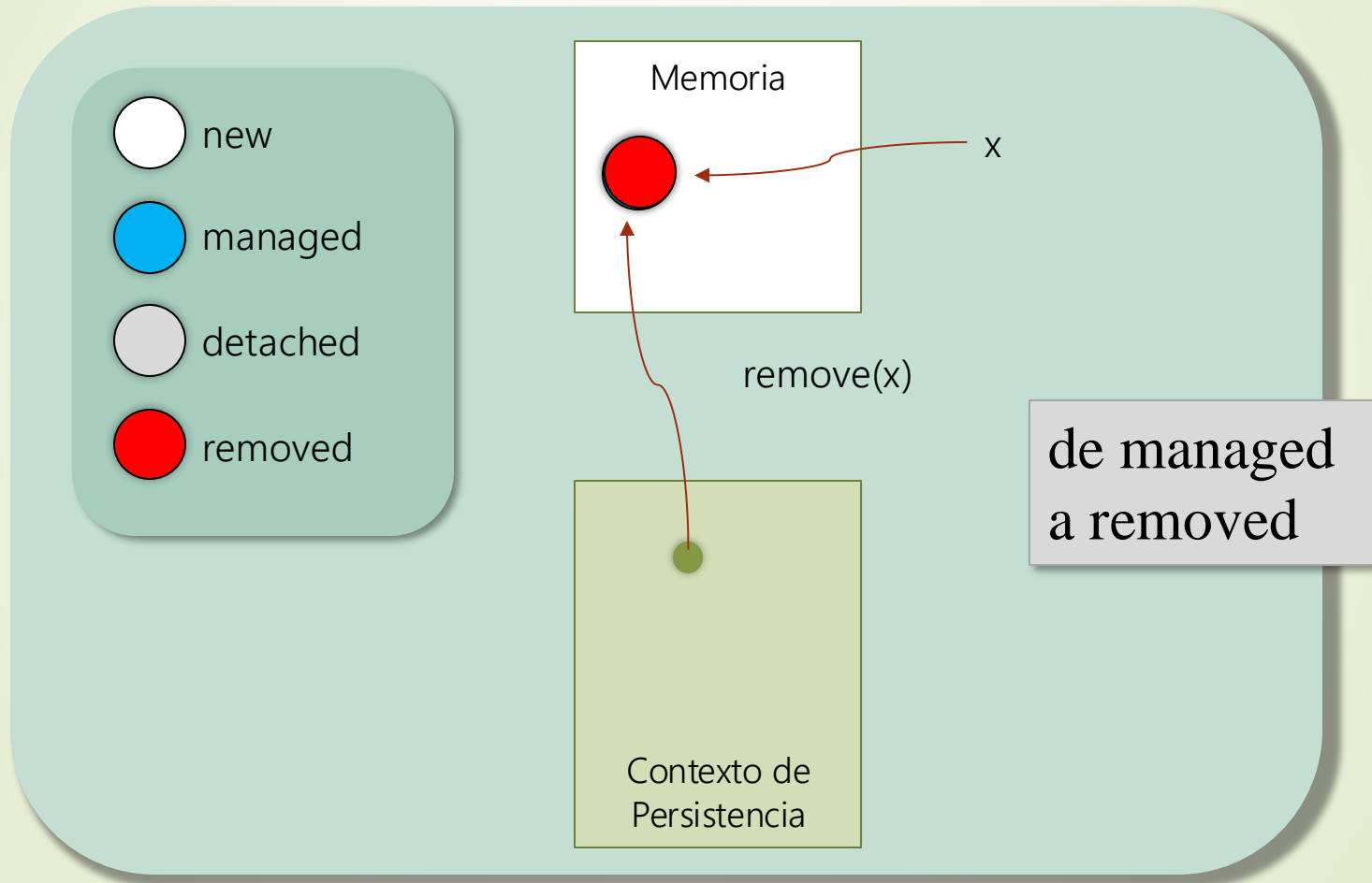
merge()

Combina el estado de una instancias con el contexto



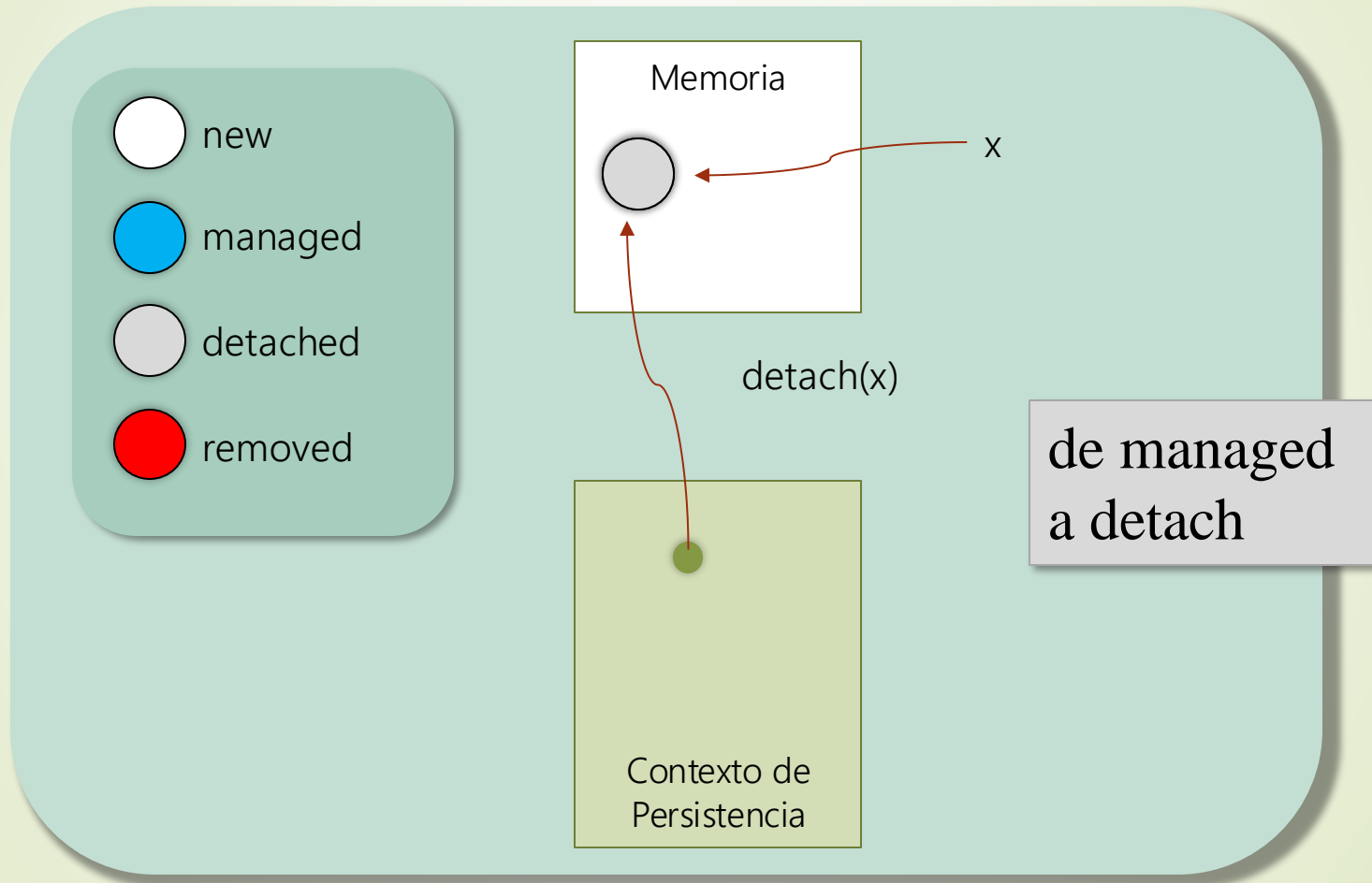
remove()

Pasa el obj de managed a removed



detach()

Desasocia la instancia del contexto



JPA – Si los nombres no coinciden

<<Java Class>>	
Persona	
(default package)	
idPersona: int	
apellidos: String	
apodo: String	
nombre: String	
dni: String	
Persona()	
getIdPersona():int	
setIdPersona(int):void	
getApellidos():String	
setApellidos(String):void	
getApodo():String	
setApodo(String):void	
getNombre():String	
setNombre(String):void	
getDni():String	
setDni(String):void	

```
@Entity
@Table(name="personas")
public class Persona implements Serializable {

    @Id
    @Column (name = "id_persona")
    private int idPersona;

    @Column (name = "p_apellidos")
    private String apellidos;

    @Column (name = "p_apodo")
    private String apodo;

    @Column (name = "p_nombre")
    private String nombre;

    @Column (name = "p_dni")
    private String dni;

    public Persona() {
    }
    ...
}
```

JPA – Si los nombres no coinciden

<<Java Class>>	
Persona (default package)	
idPersona: int	
apellidos: String	
apodo: String	
nombre: String	
dni: String	
Persona()	
getIdPersona():int	
setIdPersona(int):void	
getApellidos():String	
setApellidos(String):void	
getApodo():String	
setApodo(String):void	
getNombre():String	
setNombre(String):void	
getDni():String	
setDni(String):void	

personas	
id_persona	INT(11)
p_apellidos	VARCHAR(45)
p_apodo	VARCHAR(25)
p_nombre	VARCHAR(45)
p_dni	VARCHAR(15)
Indexes	

```
@Entity
@Table(name="personas")
public class Persona implements Serializable {

    @Id
    @Column (name = "id_persona")
    private int idPersona;

    @Column (name = "p_apellidos")
    private String apellidos;

    @Column (name = "p_apodo")
    private String apodo;

    @Column (name = "p_nombre")
    private String nombre;

    @Column (name = "p_dni")
    private String dni;

    public Persona() {
    }
    ...
}
```

Otras anotaciones

@Transient

Los atributos transient no se almacenan en la bbdd

```
@Transient  
private int edad;
```

@Basic (optional=false)

El atributo no admite nulos

```
@Column (name = "p_dni")  
@Basic (optional=false)  
private String dni;
```

Tipos enumerados

@Enumerated

```
enum Genero {HOMBRE, MUJER};  
...  
private Genero sexo;
```

```
@Column (name = "p_sexo")  
private Genero sexo;
```

```
@Enumerated  
@Column (name = "p_sexo")  
private Genero sexo;
```

Almacena el orden del valor
(cuidado con el cambio de orden)

```
@Enumerated (EnumType.STRING)  
@Column (name = "p_sexo")  
private Genero sexo;
```

Almacena una cadena

Tipos Insertables

No tienen identidad. Serán persistidos dentro de un Entidad u otro insertable

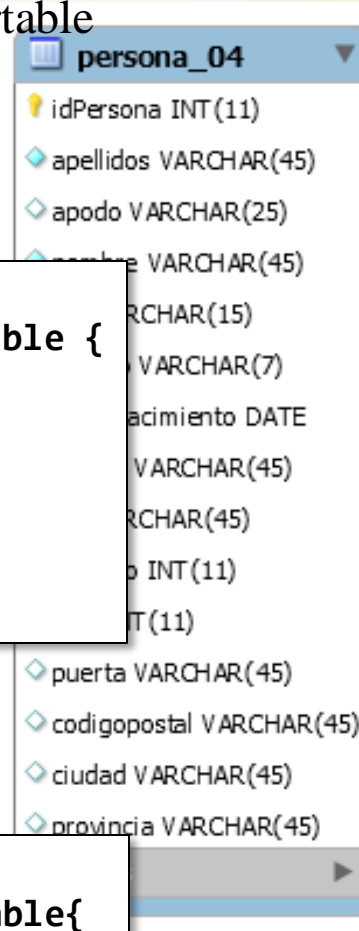
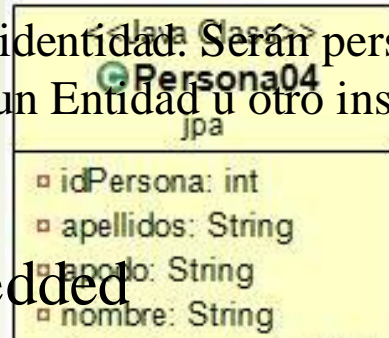
@Embedded

```
@Entity
public class Persona04 implements Serializable {
    @Id
    private int idPersona;

    @Embedded
    private Domicilio domicilio;
```

@Embeddable

```
@Embeddable
public class Domicilio implements Serializable{
```



Tipos Insertables

No tienen identidad. Serán persistidos dentro de un Entidad u otro insertable

@Embedded

```
@Entity
public class Persona04 implements Serializable {
    @Id
    private int idPersona;

    @Embedded
    private Domicilio domicilio;
```

@Embeddable

```
@Embeddable
public class Domicilio implements Serializable{
```

<<Java Class>>	
G Persona04	
jpa	
idPersona	int
apellidos	String
apodo	String
nombre	String
fechaNacimiento	Date
edad	int
dni	String
genero	Genero
domicilio	Domicilio

<<Java Class>>	
G Domicilio	
jpa	
tipoVia	String
via	String
numero	int
piso	int
puerta	String
codigoPostal	String
ciudad	String
provincia	String

persona_04	
idPersona	INT(11)
apellidos	VARCHAR(45)
apodo	VARCHAR(25)
nombre	VARCHAR(45)
dni	VARCHAR(15)
genero	VARCHAR(7)
fechanacimiento	DATE
tipovia	VARCHAR(45)
via	VARCHAR(45)
numero	INT(11)
piso	INT(11)
puerta	VARCHAR(45)
codigo postal	VARCHAR(45)
ciudad	VARCHAR(45)
provincia	VARCHAR(45)
Indexes	

PK Compuestas

- Debe crearse una clase que represente la PK, con los atributos de la PK Compuesta

PK COMO INSERTABLE

EmbeddedId property

- La PK compuesta se modela en una clase, que debe ser serializable y definida como @Embeddable
- En la Entidad, el atributo PK se declara con @EmbeddedId

MÚLTIPLES @Id

@IdClass(nombreClaseId.class)

- Múltiples atributos @Id en la clase

Tipo de Acceso

Atributos
FIELD

```
@Entity
public class Persona implements Serializable {

    @Id
    private int idPersona;

    private String apellidos;
```

Métodos
getters
PROPERTY

```
@Entity
public class Persona implements Serializable {

    private int idPersona;
    private String apellidos;

    ...

    @Id
    public int getIdPersona() {
    }
}
```

Tipos de Acceso heredado

- Las clases insertables heredan el tipo de acceso
- Se puede invalidar (cambiar) el tipo de acceso por defecto o heredado (clases Embeddable), a nivel de clase o a una propiedad en particular
- `@Access(AccessType.PROPERTY)`
- `@Access(AccessType.FIELD)`

Mapeo de Herencia

@Inheritance

➤ Single Table Strategy

@Inheritance (strategy=InheritanceType.SINGLE_TABLE)

➤ Joined Subclass Strategy

@Inheritance (strategy=InheritanceType.JOINED)

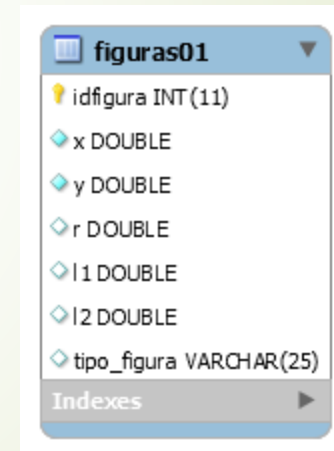
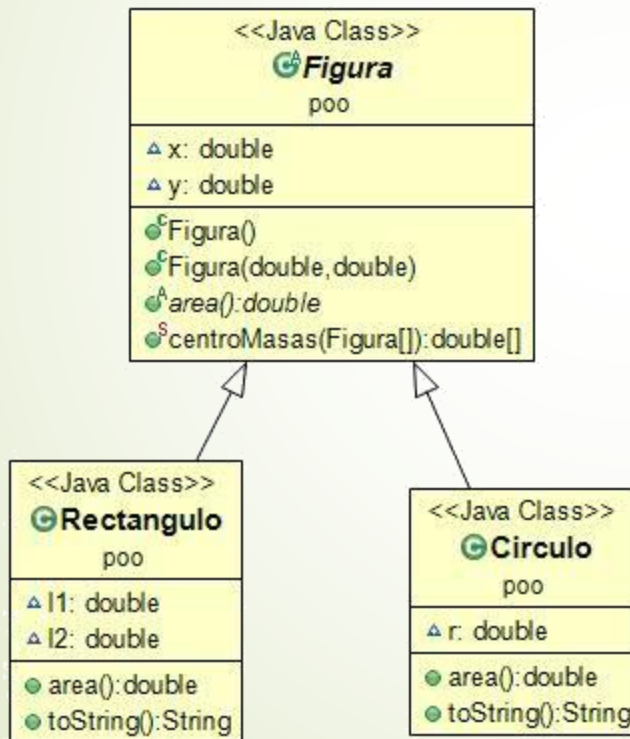
➤ Table per Class Strategy

@Inheritance (strategy=InheritanceType.TABLE_PER_CLASS)

Herencia Single Table Strategy

@Inheritance (strategy=InheritanceType.SINGLE_TABLE)

- Mapeo por defecto.
- Todas las clases de una misma familia son almacenadas en una única tabla.
- Una columna por cada atributo de cada clase y subclase de la familia.
- Una columna adicional para el tipo de clase al que hace referencia cada fila.



Herencia

Single Table Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="tipo_figura", discriminatorType=DiscriminatorType.STRING)
public abstract class Figura {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int idFigura;
    double x;
    double y;
    ...
}
```

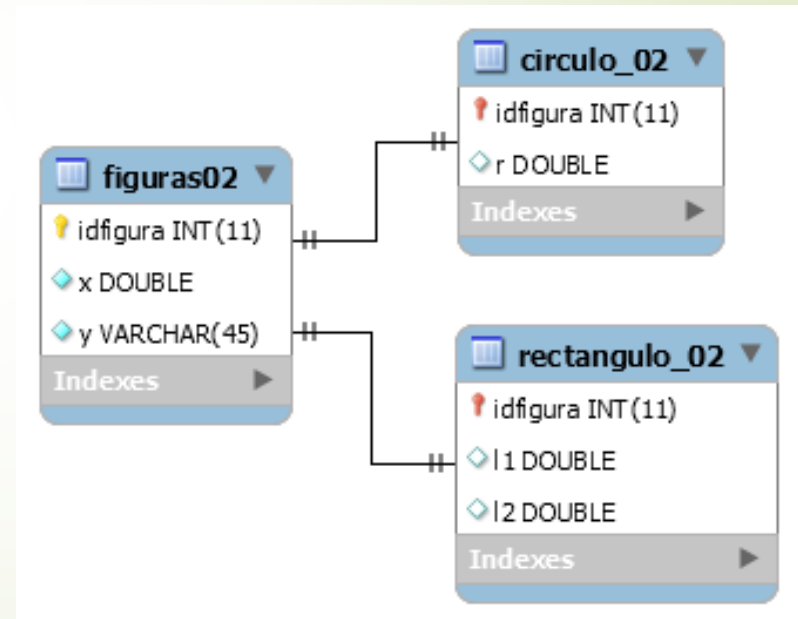
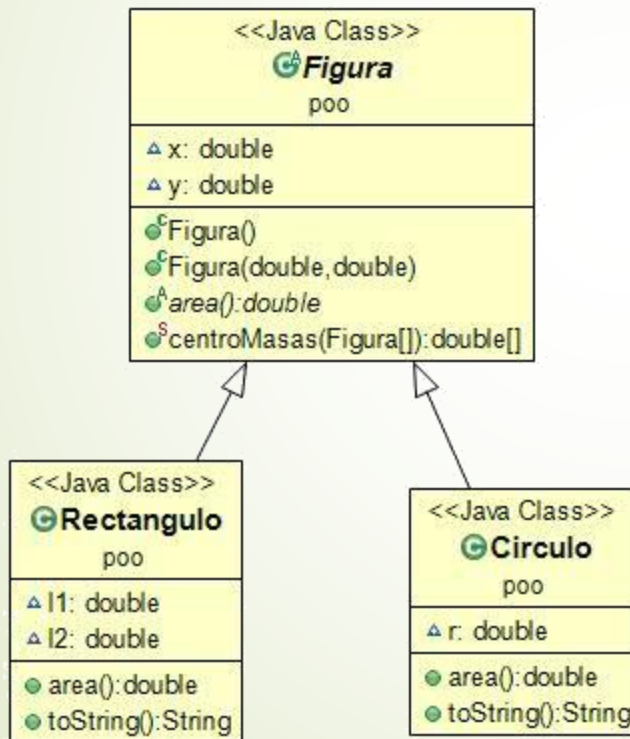
```
@Entity
@DiscriminatorValue("CIRCULO")
public class Circulo extends Figura {
    double r;
    ...
}
```

```
@Entity
@DiscriminatorValue("RECTANGULO")
public class Rectangulo extends Figura {
    double l1;
    double l2;
    ...
}
```

Herencia Joined Subclass Strategy

@Inheritance (strategy=InheritanceType.JOINED)

- Cada clase y subclase (abstracta o concreta) se almacenará en su propia tabla.
- La clave primaria de la tabla raíz es la misma que la de las subclases.
- Cada subclase almacenará en su propia tabla únicamente sus atributos propios y su clave primaria es a su vez clave foránea a su tabla raíz.



Herencia

Joined Subclass Strategy

```
@Entity
@Table (name = "figuras02")
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Figura02 {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int idFigura;

    ...
}
```

```
@Entity
@Table (name = "circulo_02")
public class Circulo02 extends Figura02 {
    double r;
}
```

```
@Entity
@Table (name = "rectangulo_02")
public class Rectangulo02 extends Figura02 {

    double l1;
    double l2;
}
```




Herencia

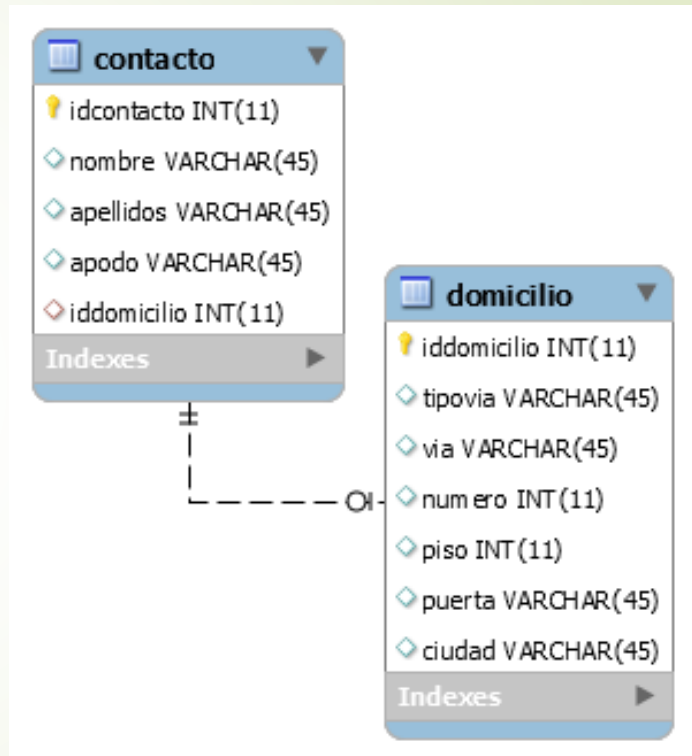
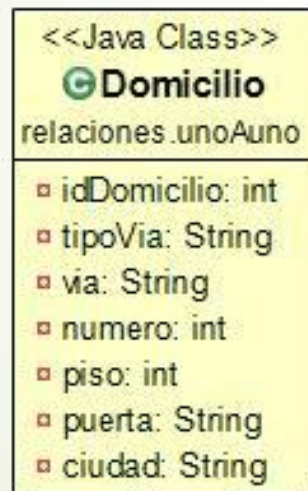
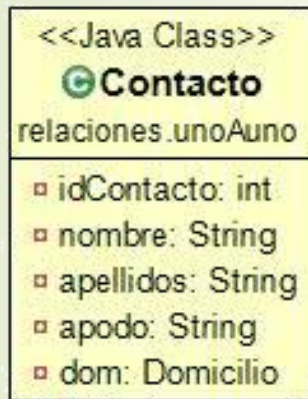
Table per Class Strategy

@Inheritance (strategy=InheritanceType.TABLE_PER_CLASS)

- Una tabla por cada clase.
- Todas las tablas tienen “todos” los atributos, propios y heredados.
- Las tablas, comparte la PK con la tabla de la clase padre.
- NO ESTÁ RECOMENDADO SU USO

Mapeo de Relaciones

(Uno a Uno)



Uno a Uno (unidireccional)

```
@Entity
public class Contacto{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int idContacto;
private String nombre;
...

@OneToOne
@JoinColumn(name = "iddomicilio")
private Domicilio dom;
```

```
@Entity
public class Domicilio {
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int idDomicilio;

private String tipoVia;
...
```

Uno a Uno (bidireccional)

```
@Entity
public class Contacto2{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int idContacto;
private String nombre;
...

@OneToOne
@JoinColumn(name = "iddomicilio")
private Domicilio dom;
```

```
@Entity
@Table (name = "domicilio")
public class Domicilio2 {

@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int idDomicilio;

private String tipoVia;
...
@OneToOne(mappedBy = "dom")
private Contacto2 contacto;
```

Tipos básicos (Colecciones)

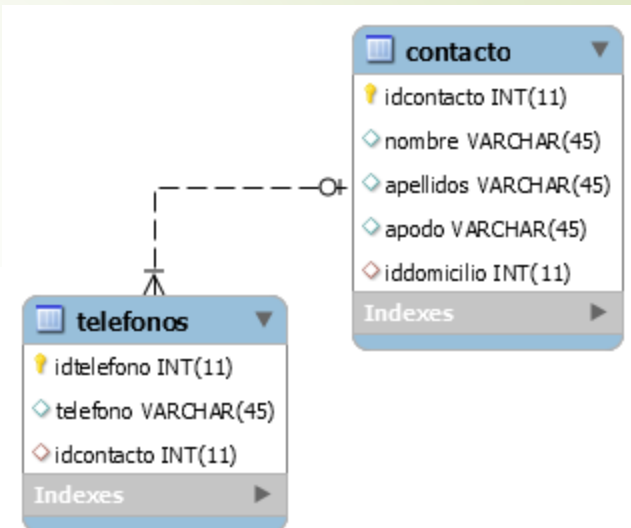
No son relac. Muchos a uno



```
@Entity
@Table (name = "contacto")
public class Contacto3{
```

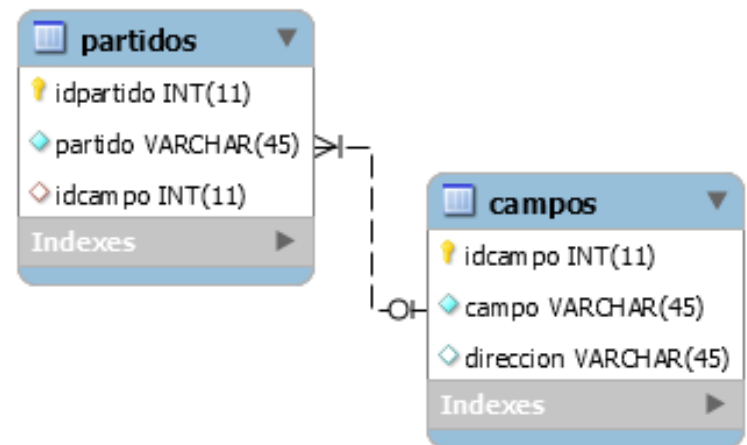
```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int idContacto;
private String nombre;
```

```
...
@ElementCollection
@CollectionTable (name="telefonos", joinColumns = @JoinColumn (name = "idcontacto"))
@Column(name="telefono")
private List<String> telefonos;
```



Muchos a Uno

unidireccional



Muchos a Uno

unidireccional

```
@Entity
@Table(name="partidos")
public class Partido implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idpartido;
    private String partido;

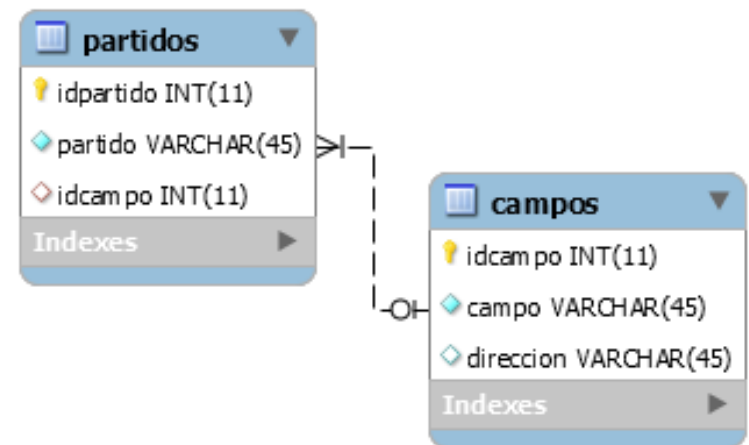
    @ManyToOne
    @JoinColumn (name = "idcampo")
    private Campo campo;
```

```
@Entity
@Table(name="campos")
public class Campo implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idcampo;
    private String campo;
    private String direccion;
```


Muchos a Uno

unidireccional



Muchos a Uno

bidireccional

```
@Entity
@Table(name="partidos")
public class Partido implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idpartido;
    private String partido;

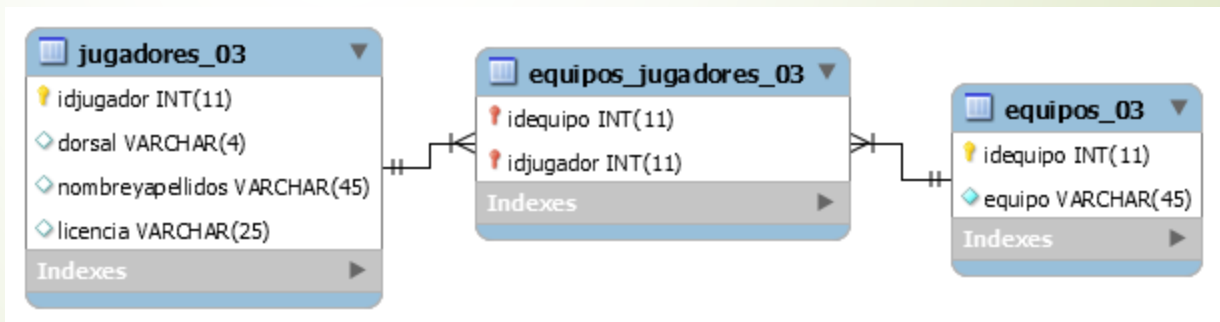
    @ManyToOne
    @JoinColumn (name = "idcampo")
    private Campo campo;
```

```
@Entity
@Table(name="campos")
public class Campo implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idcampo;
    private String campo;
    private String direccion;

    @OneToMany (mappedBy="campo")
    private List<Partido> partidos;
```

Muchos a Muchos



Muchos a Muchos

```
@Entity
@Table(name="equipos_03")
public class Equipo03 implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idEquipo;
    private String equipo;

    @ManyToMany
    @JoinTable(name="equipos_jugadores_03"
        , joinColumns={ @JoinColumn(name="IdEquipo")}
        , inverseJoinColumns={@JoinColumn(name="idJugador")})
    private List<Jugador03> jugadores;
```

```
@Entity
@Table(name="jugadores_03")
public class Jugador03 implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idJugador;
    private String dorsal;
    private String nombreYApellidos;
    private String licencia;

    @ManyToMany(mappedBy="jugadores")
    private List<Equipo03> equipos;
```



Lectura Temprana y Demorada – Fetch Type

LAZY - Lectura Demorada

Leerá los datos de la tabla en el momento que quiera utilizarse la propiedad

@ElementCollection

@OneToMany

@ManyToMany

EAGER - Lectura Temprana

Leerá los datos de la tabla al leer el objeto

@OneToOne

@ManyToOne

```
@Basic (fetch = FetchType.LAZY)  
private Imagen img;
```

```
@OneToOne(fetch = FetchType.LAZY)
```

```
@ElementCollection(fetch=FetchType.EAGER)
```

JP QL

- Permite realizar consultas sobre **Objetos** (no tablas)
- Sintaxis similar a SQL

SELECT c FROM Contacto c

```
Query q = em.createQuery("SELECT c FROM Contacto c");  
List<Contacto> lista = q.getResultList();
```

- Obtiene todas las instancias de la clase Contacto
- Contacto es el nombre de la clase (no de la tabla)
- c es una abreviatura de la Clase y con él puedo acceder a los atributos de la Clase como si fuera una instancia

SELECT c.nombre FROM Contacto c

```
Query q = em.createQuery("SELECT c.nombre FROM Contacto c");  
List<String> lista = q.getResultList();
```

JP QL

➤ DISTINCT

➤ `SELECT DISTINCT c.nombre FROM Contacto c`

➤ COUNT()

➤ `SELECT COUNT (c) FROM Contacto c`

➤ WHERE

➤ `SELECT c FROM Contacto C WHERE condiciones`

➤ BETWEEN

➤ `... WHERE c.valor BETWEEN 100 AND 1000`

➤ LIKE (% y _)

➤ `... WHERE c.nombre LIKE '%bravo%'`

➤ ORDER BY

➤ `SELECT c FROM Contacto c ORDER BY c.apellidos`

JP QL

Queries con parámetros

► Por posición

- ...WHERE c.idContato = ?1 AND c.algo = ?2
- q.setParameter(1, 107)
- q.setParameter(2, "valor de algo")

► Por nombre

- ...WHERE c.idContato = :id AND c.algo = :algo...
- q.setParameter("id", 107)
- q.setParameter("algo", "su valor")



JP QL

Actualización

➤ UPDATE

➤ UPDATE Contacto c SET c.nombre = 'Pedro'
WHERE c.IdContacto = 105

➤ DELETE

➤ DELETE FROM Contacto c WHERE c.idContacto = 107

JP QL

Ejecución

➤ **createQuery(String jpql) Query**

```
Query q = em.createQuery("SELECT c FROM Contacto c");  
List<Contacto> lista = q.getResultList();
```

➤ **createQuery(String jpql, Class<T>) TypedQuery**

```
TypedQuery<Contacto> q = em.createQuery(  
    "SELECT c FROM Contacto c WHERE c.nombre LIKE :nombre",  
    Contacto.class);  
q.setParameter("nombre", "%" + nombre + "%");  
List<Contacto> lista = q.getResultList();
```

JP QL

NamedQueries

➤ **createNamedQuery(String jpql) (Estáticas / Con Nombre)**

```
@Entity
```

```
@Table(name = "contactos")
```

```
@NamedQuery(name = "Contacto.findAll", query = "SELECT c  
FROM Contacto c")
```

```
public class Contacto implements Serializable {
```

```
    ...
```

```
}
```

```
TypedQuery q = em.createNamedQuery("Contacto.findAll",  
    Contacto.class);
```

```
List<Contacto> lista = q.getResultList();
```

JP QL

Native SQL Queries

```
Query query = em.createNativeQuery("SELECT * FROM CONTACTOS",  
Contacto.class);  
List<Contacto> lista = query.getResultList();
```

```
Query query = em.createNativeQuery("DELETE FROM CONTACTOS");  
int filas = query.executeUpdate();
```