



CURSO DE **HIBERNATE 5**


OpenWebinars



HIBERNATE

(2) HIBERNATE



Más que un ORM.
Comparativa con
otros productos. JPA.
Maven. Módulos



(4) ENTIDADES

Definición del modelo
del dominio. Entidades
y ciclo de vida. XML y
anotaciones. Tipos de
datos.



(1) INTRODUCCION

Persistencia, desfase
objeto-relacional,
ORM. Productos y
estándares



(3) PRIMER PROYECTO

Hibernate.cfg.xml,
EntityManager y
persistence.xml



(5) ASOCIACIONES

ManyToOne, OneToMany,
OneToOne, ManyToMany



HIBERNATE





HIBERNATE

(12) ENVERS



Introducción a la
auditoria de entidades.



(11) CONSULTAS HPQL VS JPQL

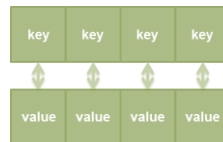
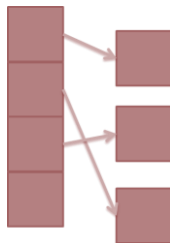
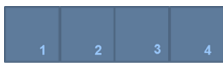
Consultas con
parámetros,
Anotaciones. SQL nativo



1.

**COLECCIONES DE
VALORES BÁSICOS
@ElementCollection**

COLECCIONES DE VALORES BÁSICOS



Bag

- Lineal
- Sin posibilidad de orden
- Con repetidos

List

- Lineal
- Posibilidad de orden.
- Con repetidos

Set

- No soporta duplicados.
- Orden con *SortedSet*

Map

- Estructura clave, valor.
- Orden con *SortedMap*

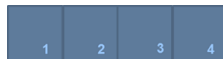
BAG (Lista sin orden)



- ▶ `Collection<E> bag = new ArrayList<E>()`
- ▶ No tiene orden
- ▶ Muy ineficiente en el borrado.

```
@ElementCollection
@CollectionTable(name="PhonesBags", joinColumns=@JoinColumn(name="person_id"))
//@OrderColumn es ignorada en la práctica
private Collection<String> phones = new ArrayList<String>();
```

LIST (Lista con posibilidad de orden)



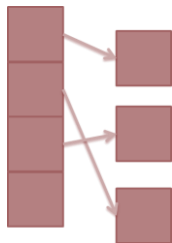
A diagram representing a list with four ordered elements. It consists of four adjacent rectangular boxes, each containing a number from 1 to 4, indicating the sequence of the elements.

1	2	3	4
---	---	---	---

- ▶ `List<E> list = new ArrayList<E>()`
- ▶ Con orden si añadimos `@OrderColumn`

```
@ElementCollection
@CollectionTable(name="PhonesList", joinColumns=@JoinColumn(name="person_id"))
@OrderColumn(name="order_id")
private List<String> phones = new ArrayList<String>();
```

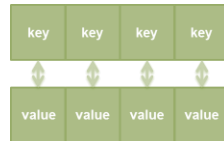

SET (Conjuntos)



- ▶ `Set<E> set = new HashSet<E>()`
- ▶ No permite repetidos
- ▶ Si queremos orden (en memoria) debemos usar `SortedSet<E> set = TreeSet<E>()` (nativo de Hibernate).

```
@ElementCollection
@CollectionTable(name="PhonesSet", joinColumns=@JoinColumn(name="person_id"))
private Set<String> phones = new HashSet<String>();
```

MAP (Clave, Valor)



- ▶ `Map<K,V> map = new HashMap<K, V>()`
- ▶ Pares *clave, valor*.
- ▶ Si queremos orden (en memoria) debemos usar `SortedMap<K, V> map = TreeMap<K,V>();` (Hibernate nativo)

```
@ElementCollection
@CollectionTable(name="PhonesMap", joinColumns=@JoinColumn(name="person_id"))
private Map<String, String> phones = new HashMap<String, String>();
```



2.

COLECCIONES DE TIPOS EMBEDD

COLECCIONES DE TIPOS EMBEDD

- Funcionan igual que con los valores básicos.

```
@ElementCollection  
private List<Phone> phones = new ArrayList<Phone>();
```



3.

COLECCIONES DE ENTIDADES

COLECCIONES DE ENTIDADES

- ▶ Ya hemos trabajado con ellas en las asociaciones **@OneToMany** y **@ManyToMany**.
- ▶ Podemos usar cualquier colección de las que hemos visto antes.
- ▶ Podemos usar **@OrderBy** y **@OrderColumn**

```
@OneToMany(cascade = CascadeType.ALL)
@OrderBy("number")
private List<Phone> phones = new ArrayList<Phone>();
```

```
@OneToMany(cascade = CascadeType.ALL)
@OrderColumn(name = "order_id")
private List<Phone> phones = new ArrayList<>();
```