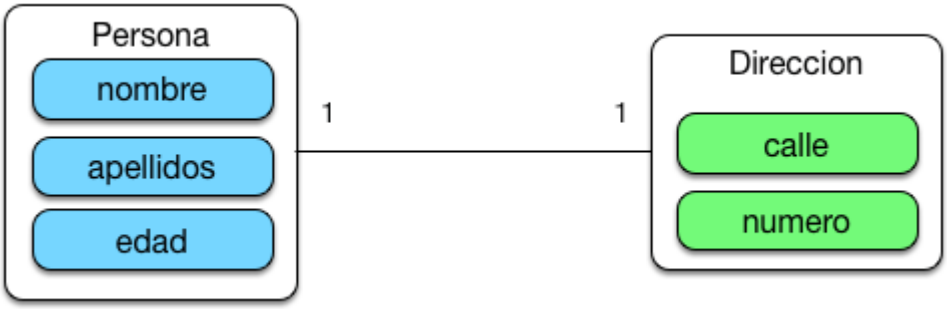
¿Qué es JPA embedded objects? . Una de las características fundamentales de los modelos de dominio es que no tienen la misma estructura que los modelos E-R de las bases de datos y permiten crear nuevos tipos de entidades y mapearlas de formas muy diversas. Vamos a construir un ejemplo sencillo de como usar las anotaciones de JPA embedded para crear un modelo diferente al de la base de datos. Partiremos de una tabla Persona con los siguientes campos.



JPA Embedded y Anotaciones

Aunque disponemos de una sola tabla en la base de datos , existen dos conceptos asociados a la información. El concepto de Persona y el concepto de Dirección.



Podemos definir nuestro modelo de clases de tal forma que mapeemos ambas clases contra la misma tabla utilizando las anotaciones de JPA Embedded y Embeddable.

```
package com.arquitectajava;

import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {
@Id
private String nombre;
private String apellidos;
private int edad;
@Embedded
private Direccion direccion;

public Persona() {
super();
}

public Persona(String nombre, String apellidos, int edad, String calle
, int numero) {
super();
this.nombre = nombre;
this.apellidos = apellidos;
this.edad = edad;
this.direccion = new Direccion(calle,numero) ;
}

public String getApellidos() {
return apellidos;
}
```

```
public String getCalle() {
return direccion.getCalle();
}

public Direccion getDireccion() {
return direccion;
}

public int getEdad() {
return edad;
}

public String getNombre() {
return nombre;
}

public int getNumero() {
return direccion.getNumero();
}
public void setApellidos(String apellidos) {
this.apellidos = apellidos;
}

public void setCalle(String calle) {
direccion.setCalle(calle);
}

public void setDireccion(Direccion direccion) {
this.direccion = direccion;
}
```

```
public void setEdad(int edad) {
this.edad = edad;
}

public void setNombre(String nombre) {
this.nombre = nombre;
}

public void setNumero(int numero) {
direccion.setNumero(numero);
}

}
```

Hemos usado la anotación @Embedded para añadir dentro la clase Persona una relación hacia el concepto de Dirección. El siguiente paso es anotar la clase Dirección con la anotación @Embeddable

```
package com.arquitectajava;

import javax.persistence.Embeddable;

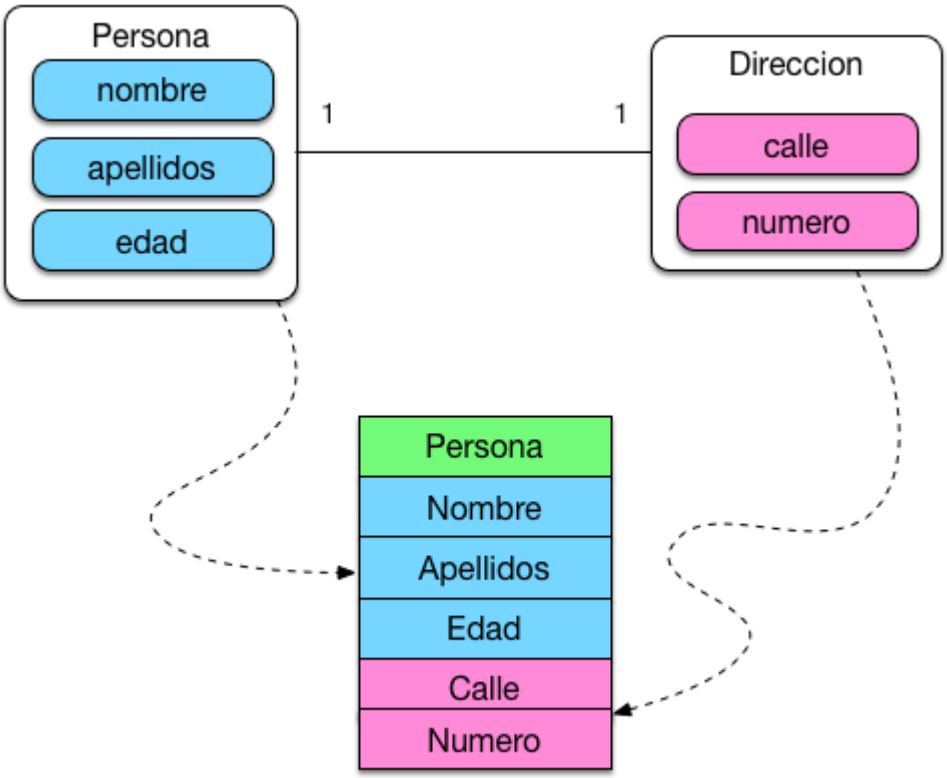
@Embeddable
public class Direccion {

private String calle;
private int numero;
public String getCalle() {
return calle;
```

```
}
public void setCalle(String calle) {
this.calle = calle;
}
public int getNumero() {
return numero;
}
public void setNumero(int numero) {
this.numero = numero;
}
public Direccion(String calle, int numero) {
super();
this.calle = calle;
this.numero = numero;
}
public Direccion() {
super();
}

}
```

Acabamos de relacionar dos clases con una única tabla de la base de datos . De esta forma podremos crear modelos más flexibles.



Una vez construido el modelo podemos salvar nuestros objetos en la base de datos:

```
package com.arquitectajava;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.FlushModeType;
import javax.persistence.Persistence;

public class Principal {
```

```
public static void main(String[] args) {

EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadBlog");
EntityManager em = emf.createEntityManager();
EntityTransaction transaccion = em.getTransaction();

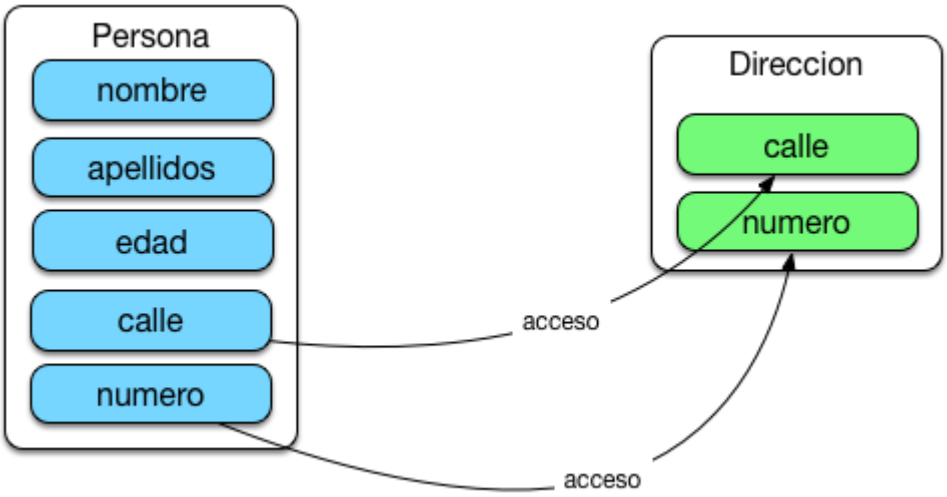
transaccion.begin();

Persona p1 = new Persona("maria", "sanchez", 20 ,"micalle",5);
em.persist(p1);
transaccion.commit();
em.close();

}

}
```

Hemos usado el concepto de delegación para generar campos que luego use la clase dirección internamente.



Podemos acceder a su valor o directamente o a través de composición:

```
package com.arquitectajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal2 {

public static void main(String[] args) {

EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadBlog");
```

```
EntityManager em = emf.createEntityManager();
TypedQuery<Persona> consulta=em.
createQuery("select p from Persona p", Persona.class);
List<Persona> lista=consulta.getResultList();

for (Persona e: lista) {
System.out.println(e.getDireccion().getCalle());
System.out.println(e.getDireccion().getNumero());
}
em.close();
}

}
```

La consulta nos imprimirá los datos que se han almacenado en la dirección.

```
may 31, 2016 12:54:20 PM org.hibernat
INFO: HHH000115: Hibernate connection
Tue May 31 12:54:20 CEST 2016 WARN: E
may 31, 2016 12:54:20 PM org.hibernat
INFO: HHH000400: Using dialect: org.h
may 31, 2016 12:54:21 PM org.hibernat
INFO: HHH000397: Using ASTQueryTransl
Hibernate: select persona0_.nombre as
micalle
5
```

Tendremos una única tabla en la base de datos pero dos clases en el modelo .Podemos usar el concepto de delegación para acceder directamente a ellos.

```
package com.arquitectajava;
```

```
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal2 {

public static void main(String[] args) {

EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadBlog");
EntityManager em = emf.createEntityManager();
TypedQuery&lt;Persona&gt; consulta=em.
createQuery("select p from Persona p", Persona.class);
List&lt;Persona&gt; lista=consulta.getResultList();

for (Persona e: lista) {
System.out.println(e.getCalle());
System.out.println(e.getNumero());
}
em.close();
}

}
```

De esta forma podremos reutilizar la clase Dirección en otras partes de nuestro modelo.

Otros artículos relacionados: JPA orm.xml , JPA Named Queries , JPA MongoDB ,Oracle JPA