Asociaciones OneToMany

Elementos avanzados en tu API REST con Spring Boot

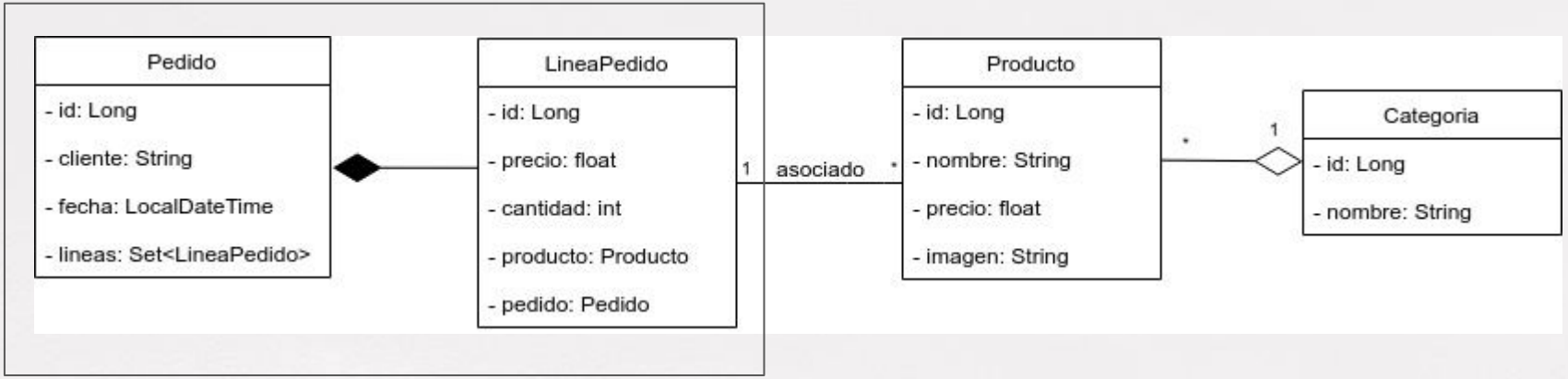
Asociación uno a muchos

- Suele ser **complementaria** a una asociación *muchos a uno*.
- Los expertos en JPA/Hibernate no recomiendan implementar asociaciones uno-a-muchos unidireccionales, por el mal rendimiento que provocan.
    - Ejemplo: la asociación *Producto* ← *Categoría* generaría 3 tablas: una para los *productos*, otra para *categorías*, y una tabla *join*.
- Siempre es preferible el tratamiento **bidireccional**, que proporciona un mejor rendimiento.

Bidireccionalidad

- Recomendable para el tratamiento eficiente de la asociación.
- Sin embargo, con Lombok podemos tener problemas de recursión infinita.
- Para solucionarlo, necesitamos algunos elementos

En nuestro ejemplo



¿Qué necesitamos para que la bidireccionalidad no provoque error?

- A nivel de entidad, con lombok

```
// Anotaciones
public class Pedido {

	@EqualsAndHashCode.Exclude
      @ToString.Exclude
	@Builder.Default
	@OneToMany(mappedBy = "pedido",
	cascade = CascadeType.ALL,
	orphanRemoval = true)
	private Set<LineaPedido> lineas =
			new HashSet<>();

	// resto de atributos y métodos
}
```

```
// Anotaciones
public class LineaPedido {

	@ManyToOne
	@JoinColumn(name = "pedido_id")
	private Pedido pedido;

	// resto de atributos y métodos
}
```

¿Qué necesitamos para que la bidireccionalidad no provoque error?

- A nivel de JSON, dos anotaciones nuevas

```
// Anotaciones
public class Pedido {

	@JsonManagedReference
		@EqualsAndHashCode.Exclude
	@ToString.Exclude
		@Builder.Default
		@OneToMany(mappedBy = "pedido",
		cascade = CascadeType.ALL,
		orphanRemoval = true)
		private Set<LineaPedido> lineas =
				new HashSet<>();

		// resto de atributos y métodos
}
```

```
// Anotaciones
public class LineaPedido {

		@JsonBackReference
	@ManyToOne
		@JoinColumn(name = "pedido_id")
		private Pedido pedido;

		// resto de atributos y métodos
}
```

¿Qué necesitamos para que la bidireccionalidad no provoque error?

- *JsonManagedReference*:
  - la propiedad anotada es parte de una asociación bidireccional
  - su función es el enlace "padre" (o "reenvío").
  - El tipo de valor (clase) de la propiedad debe tener una única propiedad compatible anotada con *JsonBackReference*.
  - Se serializa/deserializa con normalidad
  - Es la referencia inversa coincidente que requiere un manejo especial

¿Qué necesitamos para que la bidireccionalidad no provoque error?

- *JsonBackReference*:
  - la propiedad anotada es parte de una asociación bidireccional
  - su función es el enlace "hijo" (o "atrás").
  - La propiedad debe ser un *bean*, no una colección
  - No se serializa
  - En la deserialización, se crea un instancia dentro de la colección *JsonManagedReference*

¿Qué tratamiento usar por petición?

- Dependerá mucho de la semántica del problema.
- A nivel general podemos decir
  - Sigue siendo recomendable no usar entidades (sobre todo en sistemas complejos o con entidades con muchas propiedades).
  - Si el lado muchos de la asociación tiene a su vez más asociaciones, el modelo de respuesta para peticiones GET puede ser muy grande.

Petición GET (Todos, búsqueda)

- Analizar bien los atributos necesarios a mostrar
- Analizar también muy bien los atributos de la parte hija.
- Probablemente, la mejor solución sea crear un DTO (en 2 niveles, DTO para la parte padre, y otro DTO para la parte hija).

Petición GET por ID

- Analizar bien si es necesario devolver todos los atributos.
- Posiblemente, en la parte hija no sea necesario devolver los atributos que hagan relación a otras asociaciones.
- Se podría utilizar DTO en la parte hija de la relación, o también @JsonView.

Petición POST - Nuevo recurso

- Analizar bien la semántica del problema.
- Si la asociación one-to-many es de composición, probablemente la mejor solución es crear un DTO que incluya los datos de la parte padre y de la parte hija de la asociación.
- Por ejemplo:
  - *NuevoPedidoDTO*, con *nombre* y *Set<NuevaLineaPedidoDTO>*
  - *NuevaLineaPedidoDTO*, con el *ID* del producto y la *cantidad*.

Petición PUT - Modificar recurso

- Dependerá mucho de la semántica del problema (¿qué atributos sí se pueden modificar y cuáles no?
- Posiblemente necesitemos un nuevo DTO, o podamos reutilizar el DTO de la petición POST.

Petición DELETE

- Borrado de un recurso
  - Si la asociación es de composición, posiblemente necesite un *borrado en cascada*.
  - No suele necesitar representación.
  - Si va bien, se suele devolver *204 No Content*.
  - Si en alguna circunstancia, queremos devolver el recurso recién borrado, podemos usar el DTO de GET por ID.

RETO Interesante

- En esta lección nuestra aplicación ha crecido un poco (sobre todo en modelo). Se propone completar la parte del controlador, realizando algunas tareas de las lecciones anteriores.

RETO Interesante

1. Crear un DTO para las peticiones GET (todos los recursos) para Pedido y Línea de pedido. Se puede realizar la transformación en una nueva clase Converter, y usando Lombok.
2. Crear una vista con @JsonView para las peticiones GET por ID, de forma que en las líneas de venta, para el atributo producto, no se muestre el precio (lo tenemos en línea de venta), ni la imagen ni la categoría.

RETO Interesante

3. Crear la petición GET por ID en el controlador, siguiendo el ejemplo de las que ya hemos realizado anteriormente (Producto, Categoría).

4. Crear la petición POST para insertar un nuevo Pedido con sus Líneas de pedido. Para ello, vamos a crear un par de DTOs

- *NuevoPedidoDTO*, con *nombre* y *Set<NuevaLineaPedidoDTO>*
- *NuevaLineaPedidoDTO*, con el *ID* del producto y la *cantidad*.

El conversor de NuevaLineaPedidoDTO a LineaPedido tendrá que usar el servicio de producto, para transformar un ID de producto en un Producto.

RETO Interesante

5. Crear la petición PUT, que será muy similar a las POST.

6. Crear la petición DELETE, que será muy parecida a la de producto (con la asociación bidireccional y el borrado en cascada, al borrar un *Pedido*, se borrarán sus *LineaPedido* asociadas).