Uso del patrón DTO

Desarrollo de un API REST con Spring Boot

Entidades

- Para nosotros, clases Java con algunas anotaciones más.
- Cumplen una serie de requisitos.
- Se encargan de modelar nuestros objetos en la capa de lógica de negocio.
- ¿Es bueno que los usemos en el resto de capas?
    - ***Depende**, pero es buena práctica no hacerlo.*

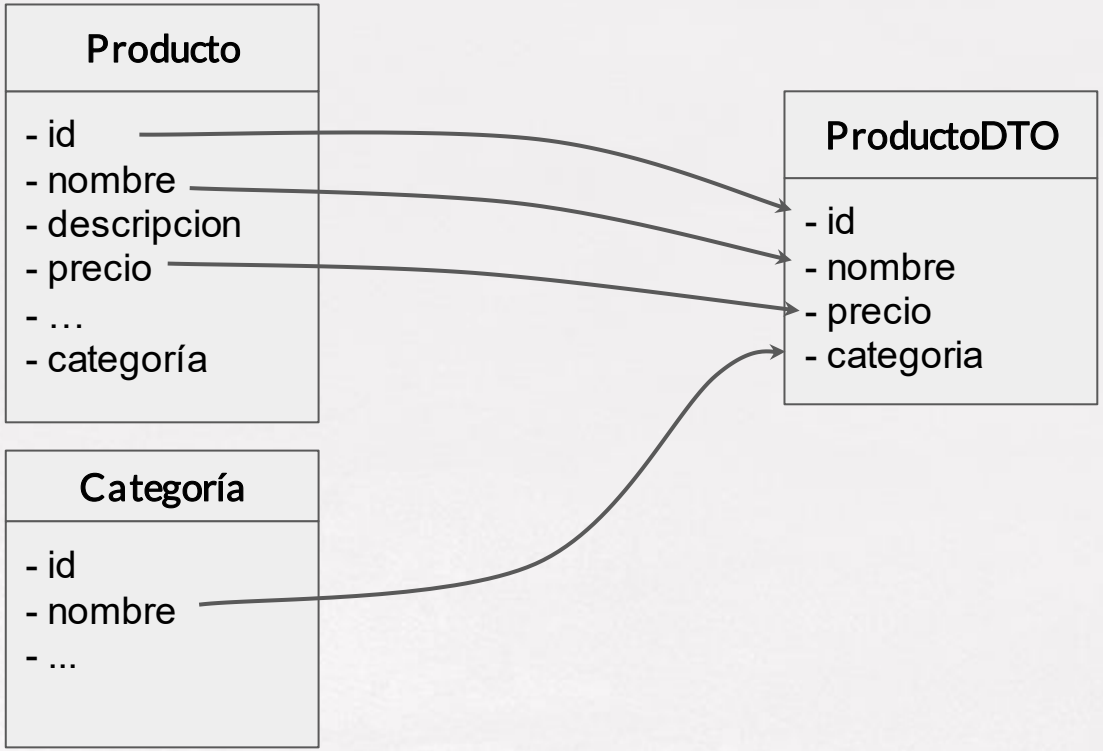
DTO: *Data Transfer Object*

- Objeto POJO que agrupa datos de la capa de negocio.
  - Puede tener parte de los datos de una sola entidad
  - Puede tener algunos datos de más de una entidad.
  - Puede aglutinar todos los datos de varias entidades.
- A veces es conocido como *Value Object*.
- Pensado para *aligerar* las transacciones entre cliente/servidor.

DTO: Características

- Objeto plano (POJO). Nada de lógica de negocio.
  - Getters, Setters o los constructores necesarios para facilitar el trabajo.
- Serializable
  - Para que pueda viajar a través de la red.

Ejemplo


Producto
- id
- nombre
- descripcion
- precio
- …
- categoría
Categoría
- id
- nombre
- ...
ProductoDTO
- id
- nombre
- precio
- categoria

Cómo y dónde usarlo

- Supongamos que tenemos un catálogo de productos como el de nuestro ejemplo.
  - GET /producto/
  - GET /producto/{id}

GET /producto/

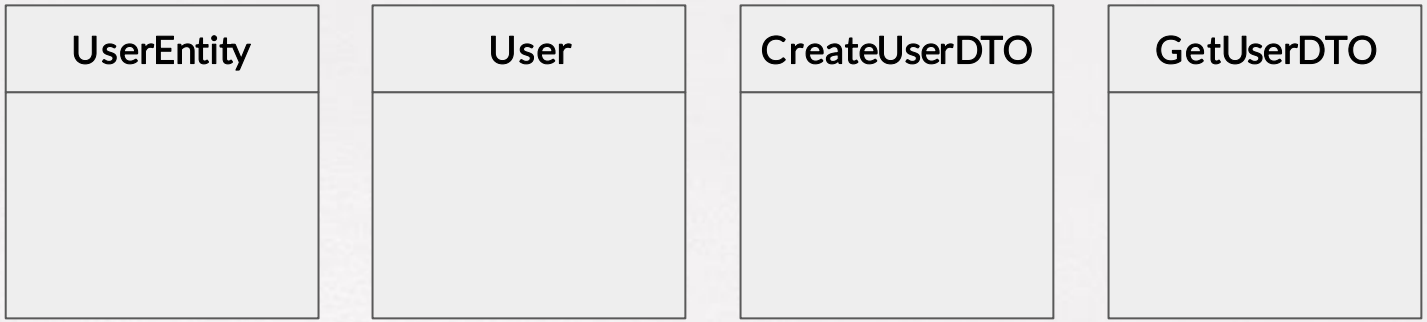
- El listado de todos los productos puede ir orientado a ser visualizado en una página con todos los productos (o un subcojunto paginado de ellos).
- ¿Necesitamos todos los datos del producto? NO
- Podemos usar un DTO que incluya solamente los datos necesarios.

GET /producto/{id}

- Nos permite obtener todos los datos de un producto en particular.
  - En este caso puede que no necesitemos usar el DTO anterior.
  - Alternativas
    - No usar DTO y usar la entidad
    - Crear otro DTO específico.

Múltiples DTOs para un solo BO

- Es posible tener más de un Data Transfer Object para un solo objeto de negocio.
- Ejemplo clásico: *Usuario*.



Defensores y detractores

- DTO es un patrón de diseño que tiene defensores y detractores.
- Cada equipo de trabajo puede encontrar su forma adecuada de transportar los datos a través de las diferentes capas de nuestra aplicación.
- Mi recomendación
  - No ofrecer más datos de los debidos en una petición.
  - No abusar del DTO creando uno para cada tipo de petición.

Cómo implementar DTO

- *Manualmente*
  - Nos encargamos de *gettear*/*settear* los datos necesarios entre los diferentes objetos.
  - Se puede crear algún *builder* (lombok)
- ModelMapper
- *JsonViews*
  - A través de anotaciones, un mismo objeto puede devolver más o menos datos.