

≡ 06

Sobre o DTO

Praticamos o recebimento de dados do formulário no nosso *controller*. É simples de fazer, basta usar o mesmo `name` do `input` HTML na classe DTO:

```
<input name="nomeProduto" placeholder="nome do produto"/>
```

Mapeado para:

```
public class RequisicaoNovoPedido {  
  
    private String nomeProduto; //get+set  
  
    //...  
}
```

Mas talvez você tenha percebido que a classe `RequisiçãoNovoPedido` é bem parecida com a classe entidade `Pedido`. A classe `Pedido` tem os mesmos atributos da `RequisiçãoNovoPedido`, adicionando apenas o valor e a data!

Ou seja, será que devemos realmente criar essa classe auxiliar? Não basta criar a classe `Pedido` apenas? Para ilustrar, vejo como poderia ficar o código do método `novo` no *controller*, usando apenas a classe `Pedido`:

```
@PostMapping("novo")  
public String novo(Pedido novoPedido) { //sem uso do DTO  
    pedidoRepository.save(novoPedido);  
  
    return "pedido/formulario";  
}
```

O código funciona e ainda fica mais simples! Qual é o problema dessa abordagem?

Selecione 2 alternativas

A Problema nenhum, uma classe a menos para manter!

B O problema é que o Spring MVC não sabe converter para tipos específicos do Java, como `BigDecimal`. Por isso devemos usar um DTO, que recebe os dados e converte para tipos específicos.

C O problema é que abrimos espaço para receber mais dados da requisição do que foi previsto. Como a classe `Pedido` têm mais atributos (o `valor` e a `data`), podemos enviar esses dados também (ao manipular os dados da requisição).

D

O problema é que expomos o nosso modelo e qualquer mudança nele causaria uma mudança no HTML.