Notes for Spring Rest - 3

✅ Full Explanation of the Concepts (Based on Your Output + DB)

You basically built a small **E-commerce Order System** that calculates the price of ordered products dynamically.

Let's break the concepts down.

🧩 1. OrderRequestDTO → Input From Client

The client sends something like:

```
[
  { "productId": 1, "quantity": 2 },
  { "productId": 6, "quantity": 3 }
]
```

This DTO only contains:

- productId
- quantity

Because the price, discount, name, stock... must be fetched from DB, not from the client.

🧩 2. Fetch Product From DB

From your `product` table:

product_id	product_name	price	discount	stock
1	iPhone 16	79999	5	12
6	Headphones	1500	15	60

So when user asks for product 1 → you fetch this row:

```
iPhone 16 price = 79999
```

When asking for product 6:

```
Headphones price = 1500
```

🧩 3. Build OrderItems for Each Product

For each product in the request:

Example 1: Product 1

- Price = 79999
- Quantity = 2

So:

```
totalProductPrice = 79999 × 2 = 159998
```

Example 2: Product 6

- Price = 1500
- Quantity = 3

```
totalProductPrice = 1500 × 3 = 4500
```

So your orderItems JSON is correct:

```
"orderItems": [
   {
      "productId": 1,
      "productName": "iPhone 16",
      "quantity": 2,
      "eachProductPrice": 79999.0,
      "totalProductPrice": 159998.0
   },
   {
      "productId": 6,
      "productName": "Headphones",
      "quantity": 3,
      "eachProductPrice": 1500.0,
      "totalProductPrice": 4500.0
   }
]
```

🧩 4. Calculate Total Amount of the Order

Add all `totalProductPrice`:

```
159998 + 4500 = 164498
```

Your output says:

```
"totalAmount": 164498.0
```

✔️ **Correct**

🧩 5. orderPrice vs totalAmount — Why orderPrice = 0?

Your JSON:

```
"orderPrice": 0.0
```

That is **correct only if you haven't implemented discount logic**.

Usually:

- **orderPrice = original total (without discount)**
- **totalAmount = price after discount or after adding tax**

You didn't apply discounts → so `orderPrice` stays 0.

Later you can implement:

```
orderPrice = sum of eachProductPrice * quantity (no discount)
totalAmount = orderPrice - discount OR + tax
```

🧩 6. cascade = CascadeType.ALL (Important!)

Because your entity probably looks like:

```
@OneToMany(mappedBy="order", cascade = CascadeType.ALL)
private List<OrderItems> orderItems;
```

That means:

- When you save **Order**, Hibernate automatically saves **OrderItems** also.
- No need to save them manually.

Perfect for your use case.

🧩 7. Final Output Explanation

Your output:

```
{
   "orderId": 1,
   "status": "Ordered",
   "orderPrice": 0.0,
   "totalAmount": 164498.0,
   "orderItems": [ ... ]
}
```

This means:

✔️ **Order saved successfully**

✔️ **OrderItems saved successfully**

✔️ **Price calculated correctly**

✔️ **Product details fetched from database**

✔️ **Cascade worked properly**

Everything is correct.

🧠 If You Want, I Can Also Explain:

✅ How Order, OrderItems, Product Entities should look

✅ Service + Repository Layer Architecture

✅ How JPA lazy loading works in this flow

✅ How to add discount logic

✅ How to validate stock before placing order

✅ How to handle OutOfStock scenario

✅ How to return an Invoice PDF

Just tell me — *"explain the entities"* or *"explain discount logic"* or anything.

I'm ready, Sai 💥