Bônus: Aprenda Fundamentos

(Spring e desenvolvimento de sistemas)

Capítulo: Modelo de domínio e ORM

https://devsuperior.com.br

DISCLAIMER

Este é um **BÔNUS** concedido aos alunos do Bootcamp Spring, sem custo adicional.

O objetivo deste bônus é ensinar os seguintes fundamentos:

- Injeção de dependência
- DTO, padrão camadas
- JPA: entidades associadas, lazy loading

Vale ressaltar também que o objetivo deste bônus **NÃO É** implementar um projeto completo, mas sim abordar os fundamentos citados acima.

Bons estudos!

Estudos prévios necessários:

Revisão Álgebra Relacional e SQL

Pra quê? Para relembrar as operações básicas com SQL.

https://www.youtube.com/watch?v=GHpE5xOxXXI

Super revisão de OO e SQL com Java e JDBC

Pra quê? Para que você compreenda na prática como é consultar os dados de um banco de dados somente com Java e JDBC, sem utilizar uma ferramenta ORM (Mapeamento Objeto-Relacional).

https://www.youtube.com/watch?v=xC_yKw3MYX4

Nivelamento ORM - JPA e Hibernate

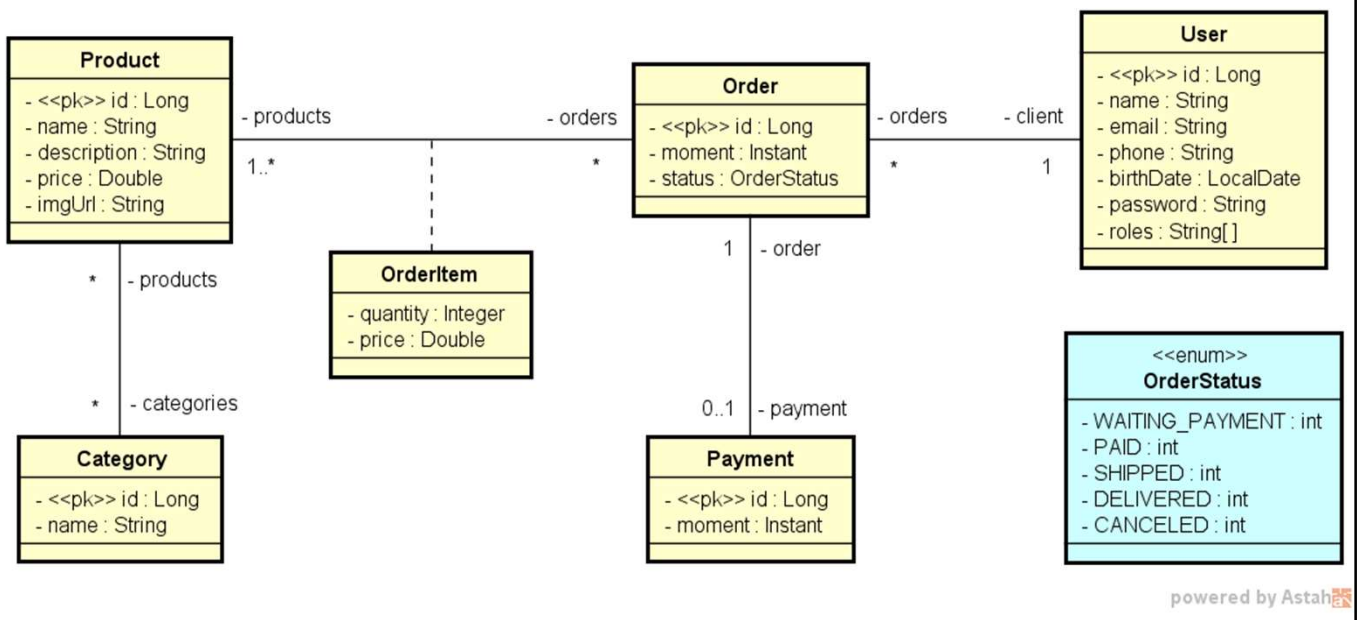
Pra quê? Para que você tenha uma introdução teórica e prática sobre ORM com JPA, **antes de ir direto para o Spring** com o Spring Data JPA.

https://www.youtube.com/watch?v=CAP1IPgeJkw

Sistema DSCommerce

Documento de requisitos:

https://drive.google.com/drive/folders/1WTBggtq38cLeeQosPHjuhjSLxa94Lmx_



Banco de dados H2, entidade User

```
spring.profiles.active=test

spring.jpa.open-in-view=false
```

```
# Dados de conexão com o banco H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

# Configuração do cliente web do banco H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Configuração para mostrar o SQL no console
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Recomendação para campo tipo Instant

```
@Column(columnDefinition = "TIMESTAMP WITHOUT TIME ZONE")
private Instant moment;
```

Caso precise aprender sobre data-hora em Java (LocalDate, LocalDateTime e Instant):
https://www.youtube.com/watch?v=WnJUI-jMQGE

Relacionamento muitos-para-um

```
public class Order {
    ...

    @ManyToOne
    @JoinColumn(name = "client_id")
    private User client;
```

```
public class User {
    ...

    @OneToMany(mappedBy = "client")
    private List<Order> orders = new ArrayList<>();
```

Relacionamento um-para-um

```
public class Payment {

    ...

    @OneToOne
    @MapsId
    private Order order;
```

```
public class Order {

    ...

    @OneToOne(mappedBy = "order", cascade = CascadeType.ALL)
    private Payment payment;
```


Relacionamento muitos-para-muitos

```
public class Product {

    ...

    @ManyToMany
    @JoinTable(name = "tb_product_category",
            joinColumns = @JoinColumn(name = "product_id"),
            inverseJoinColumns = @JoinColumn(name = "category_id"))
    private Set<Category> categories = new HashSet<>();
```

```
public class Category {

    ...

    @ManyToMany(mappedBy = "categories")
    private Set<Product> products = new HashSet<>();
```

Muitos-para-muitos com classe de associação

```
@Embeddable
public class OrderItemPK {

    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;

    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;

    ...
```

```
@Entity
@Table(name = "tb_order_item")
public class OrderItem {

    @EmbeddedId
    private OrderItemPK id = new OrderItemPK();

    private Integer quantity;
    private Double price;

    public OrderItem() {
    }

    public OrderItem(Order order, Product product, Integer quantity, Double price) {
        id.setOrder(order);
        id.setProduct(product);
        this.quantity = quantity;
        this.price = price;
    }

    public Order getOrder() {
        return id.getOrder();
    }

    public void setOrder(Order order) {
        id.setOrder(order);
    }

    ...
```


Muitos-para-muitos com classe de associação

```
public class Order {

  ...

  @OneToMany(mappedBy = "id.order")
  private Set<OrderItem> items = new HashSet<>();

  ...

  public Set<OrderItem> getItems() {
    return items;
  }

  public List<Product> getProducts() {
    return items.stream().map(x -> x.getProduct()).toList();
  }
```

```
public class Product {

  ...

  @OneToMany(mappedBy = "id.product")
  private Set<OrderItem> items = new HashSet<>();

  ...

  public Set<OrderItem> getItems() {
    return items;
  }

  public List<Order> getOrders() {
    return items.stream().map(x -> x.getOrder()).toList();
  }
```

Seeding da base de dados

- Arquivo na pasta resources: import.sql
- Utilizar um INSERT por registro
- Script:

https://gist.github.com/acenelio/664c3508edd4d418d566ed86179fdf8b

13