Desenvolvimento De Aplicações Para Web

❖ *ANNOTATIONS*

@Temporal: Tipo de data que será Persistida do Banco de Dados

Ex: @Temporal(TemporalType.TIMESTAMP): "TIMESTAMP" indica que o campo será mapeado como um tipo de data e hora completo, que inclui data e hora, com precisão até os milissegundos.

@OneToMany: Um Para muitos

CascadeType: Executa as operações em cascata, definindo como operações como inserção, atualização e exclusão devem ser propagadas automaticamente de uma entidade pai para suas entidades filhas relacionadas.

FeachType: Forma de consulta.

↳ **LAZY:** Carregamento **Preguiçoso**, onde os dados relacionados são carregados sob demanda, economizando recursos.

↳ **EAGER:** Carregamento Ansioso, onde os dados relacionados são carregados **Imediatamente** junto com a entidade principal.

```java
public class Venda {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Temporal(TemporalType.TIMESTAMP)
    private Date dataVenda;
    private Double valorTotal;
    @ManyToOne
    private Pessoa pessoa;
    @OneToMany(cascade = CascadeType.ALL,
            fetch = FetchType.EAGER,
            mappedBy = "venda")
    private List<ItensVenda> itensVendas;
    @OneToMany(cascade = CascadeType.ALL,
            fetch = FetchType.EAGER,
            mappedBy = "venda")
    private List<ContasReceber> contasRecebers;
```

MappedBy:

O uso do **"mappedBy"** ajuda o **ORM** (mapeamento objeto-relacional) a entender como as duas entidades estão conectadas e como manter a consistência do banco de dados ao persistir os objetos. Ele evita a duplicação de informações e garante que as informações estejam corretamente mapeadas entre as entidades.

AJAX (Asynchronous JavaScript and XML):

É uma técnica de programação que permite que as páginas da web atualizem conteúdo sem recarregar a página inteira. Ela é frequentemente usada para criar interfaces de usuário interativas e responsivas em aplicativos da web, permitindo a troca de dados com o servidor de forma assíncrona.

vendaedita.xhtml

```
<p:panel header="Carrinho de Compras" style="text-align: -webkit-center">
    <p:panelGrid columns="2">
        Produto:<p:autoComplete value="#{vendaControle.itensVenda.produto}"
                converter="#{vendaControle.produtoConverter}"
                completeMethod="#{vendaControle.getListaFiltrandoProduto}"
                var="prod"
                itemLabel="#{prod.nome}"
                itemValue="#{prod}"
                forceSelection="true"
                size="50">
    <p:ajax event="itemSelect"
        update="preco grupoProduto"
        listener="#{vendaControle.atualizaPreco()}"/>
        </p:autoComplete>
        Grupo do Produto:<p:inputText id="grupoProduto"
                                      value="#{vendaControle.itensVenda.produto != null ?
        vendaControle.itensVenda.produto.produtoGrupo.nome : ''}" readonly="true" size="10"/>
        Preço:<p:inputText id="preco" value="#{vendaControle.itensVenda.preco}" size="10"/>
        Quantidade:<p:inputText value="#{vendaControle.itensVenda.quantidade}" size="10"/>
        <p:commandButton value="Adicionar" action="#{vendaControle.addItem()}" ajax="false"/>
</p:panelGrid>
```

VendaControle

```
public void atualizaPreco() {
    Produto produtoSelecionado = itensVenda.getProduto();
    if (produtoSelecionado != null) {
            itensVenda.setPreco( preco: produtoSelecionado.getPreco());
            // Atualize o grupo do produto
            itensVenda.getProduto().getProdutoGrupo().getNome();
    }
}
```

Questões

1) Onde é implementado o método de Baixa de Estoque?

Facade porque ele além de salvar a venda ele faz baixa de estoque.

Certo (Pelo Facade):

O controlador chama o Facade 1 vez e ele gerencia tudo.

Errado (Por outros métodos, Ex: entidade):

O controlador chama o Facade toda vez que for fazer uma ação (salvar, registrar no estoque), ou seja mais provável de dar problema (**Ex:** Salvar e não fazer alteração do estoque, e fica aberto os outros Facades enquanto isso).

2) Saber Escrever "Public void addItem" (VendaControle)

```
public void addItem() {
    ItensVenda itemTemp = null;
    Double estoque = itensVenda.getProduto().getEstoque();
    for (ItensVenda it : venda.getItensVendas()) {
        if (itensVenda.getProduto().getId().equals( obj:it.getProduto().getId())) {
            estoque = estoque - it.getQuantidade();
            itemTemp = it;
        }
    }
    if (itensVenda.getQuantidade() > estoque) {
        FacesContext.getCurrentInstance().addMessage( clientId:null, new FacesMessage(
                severity:FacesMessage.SEVERITY_ERROR,  summary:"Estoque insuficiente!",
                "Restam apenas " + estoque + " unidade!"));
    } else {
        if (itemTemp == null) {
            itensVenda.setVenda(venda);
            venda.getItensVendas().add( e:itensVenda);
        } else {
            itemTemp.setQuantidade(itemTemp.getQuantidade() + itensVenda.getQuantidade());
        }

        itensVenda = new ItensVenda();
    }
}
```

```
public void addItem() {
```

Este é um método público que é responsável por adicionar itens a uma venda ou atualizar a quantidade de um item existente na venda.

```
    ItensVenda itemTemp = null;
```

Declaração de uma variável **itemTemp** do tipo **ItensVenda** . Inicialmente, ela é definida como **null** .

```
    Double estoque = itensVenda.getProduto().getEstoque();
```

Obtém o valor do estoque disponível para o produto associado ao item de venda atual **(itensVenda)** . O estoque é armazenado em uma variável do tipo Double.

```
    for (ItensVenda it : venda.getItensVendas()) {
```

Um loop **for-each** que percorre todos os itens de venda existentes na venda atual **(venda)** . Usado para verificar se já existe um item com o mesmo produto na venda.

```
        if
(itensVenda.getProduto().getId().equals(it.getProduto().getId())) {
            estoque = estoque - it.getQuantidade();
            itemTemp = it;
        }
    }
    if (itensVenda.getQuantidade() > estoque) {
        FacesContext.getCurrentInstance().addMessage
(null, new FacesMessage(
                FacesMessage.SEVERITY_ERROR,
"Estoque insuficiente!", "Restam apenas " + estoque + " unidade!"));
    } else {
```

Verifica se a quantidade desejada de itens a serem adicionados à venda é maior do que o estoque disponível.

- Se for maior, uma mensagem de erro é exibida, indicando que o estoque é insuficiente.
- Se for menor ou igual, o código continua a processar.

```
        if (itemTemp == null) {
```

Verifica se **itemTemp** ainda é **null** . Se for, significa que não existe um item com o mesmo produto na venda. Caso contrário, o item já existe na venda e precisa ser atualizado em vez de adicionado.

```
            itensVenda.setVenda(venda);
```

Define a venda associada ao item de venda atual.

```
            venda.getItensVendas().add(itensVenda);
```

Adiciona o item de venda à lista de itens de venda da venda atual.

```
            } else {
                itemTemp.setQuantidade(itemTemp.getQuantidade() +
itensVenda.getQuantidade());
            }
```

```
            itensVenda = new ItensVenda();
```

Após a adição ou atualização bem-sucedida do item de venda, uma nova instância de **ItensVenda** é criada, preparando o sistema para a adição de um novo item.

```
        }
    }
```