01

Introduzindo as vendas de um livro

Transcrição

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jsf-primefaces/stages/capitulo-8.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Com a nossa aplicação pronta, iremos fazer um exemplo neste capítulo de como fazer um gráfico no Primefaces. Há diversos tipos de gráficos que o Primefaces oferece, você pode vê-los [aqui][1], nós utilizaremos o [**BarChart**][2], mas vamos primeiro preparar o modelo.

Introduzindo as vendas de um livro

Por que devemos preparar mais um modelo? Porque na nossa aplicação só temos os modelos de `Livro` e `Autor`, que não são bons exemplos de dados a serem representados em um gráfico. Por isso criaremos uma nova classe que será a `Venda` justamente para representar as negociações de cada livro na forma de um gráfico *BarChart*.

Essa classe, bem simples, terá os atributos `livro` e `quantidade`, com seus *getters* e *setters* além de um construtor que recebe esses dois atributos por parâmetro:

```
public class Venda {

    private Livro livro;
    private Integer quantidade;

    public Venda(Livro livro, Integer quantidade) {
        this.livro = livro;
        this.quantidade = quantidade;
    }

    public Livro getLivro() {
        return livro;
    }
    public void setLivro(Livro livro) {
        this.livro = livro;
    }
    public Integer getQuantidade() {
        return quantidade;
    }
    public void setQuantidade(Integer quantidade) {
        this.quantidade = quantidade;
    }
}
```

Se temos um modelo, precisamos também do *bean*, já que teremos uma tela para mostrar os gráficos e cada tela no JSF precisa de um *bean*. Essa classe simulará algumas vendas, o ideal seria que as mesmas viessem do banco de dados, mas no nosso caso não há isso, então as vendas serão feitas em memória.

Teremos um método que retornará uma lista de vendas, mas como as vendas tem um livro, então precisamos pegar a lista deles também, utilizando o nosso já conhecido DAO:

```
@ManagedBean
@ViewScoped
public class VendasBean {

    public List<Venda> getVendas() {

        List<Livro> livros = new DAO<Livro>(Livro.class).listaTodos();
        List<Venda> vendas = new ArrayList<Venda>();

        return vendas;

    }
}
```

Populando as vendas

Mas por enquanto a lista de vendas está vazia. Para populá-la, vamos fazer um `foreach` nos livros, e a cada iteração nós criamos uma nova venda, adicionando-a na lista.

```
public List<Venda> getVendas() {

        List<Livro> livros = new DAO<Livro>(Livro.class).listaTodos();
        List<Venda> vendas = new ArrayList<Venda>();

        for (Livro livro : livros) {
            vendas.add(new Venda(livro, quantidade)); // o que será a quantidade?
        }

        return vendas;

    }
```

Falta criar a quantidade que será um número aleatório, vamos fazer isso utilizando a classe `Random` que recebe um `seed` por parâmetro, para sempre gerar um valor não repetido e em uma sequência. Mas como queremos sempre os mesmos valores randômicos, valores fixos, vamos fixar o valor do `seed` em 1234. Com isso, o atributo `quantidade` utilizará esse `random`, chamando o método `nextInt`, que serve para gerar um número randômico inteiro para nós, mas precisamos dizer qual será o valor máximo desse número, iremos dizer que queremos que o valor máximo de `quantidade` seja 500:

```
public List<Venda> getVendas() {

    List<Livro> livros = new DAO<Livro>(Livro.class).listaTodos();
    List<Venda> vendas = new ArrayList<Venda>();

    Random random = new Random(1234);

    for (Livro livro : livros) {
        Integer quantidade = random.nextInt(500);
        vendas.add(new Venda(livro, quantidade));
```

```
        }

        return vendas;
    }
```

Já conseguimos gerar uma lista de vendas, falta agora criar o `xhtml` e gerar o gráfico.