03

Terminando o projeto

Transcrição

Para terminar, vamos adicionar mais um `chartSerie`, representando as vendas de 2015. Basta copiar o código anterior, mudando o *label* e o nome da variável para `vendasSerie2015`:

```
public BarChartModel getVendasModel() {

    BarChartModel model = new BarChartModel();

    ChartSeries vendaSerie = new ChartSeries();
    vendaSerie.setLabel("Vendas 2016");

    List<Venda> vendas = getVendas();

    for (Venda venda : vendas) {
        vendaSerie.set(venda.getLivro().getTitulo(), venda.getQuantidade());
    }

    model.addSeries(vendaSerie);

    ChartSeries vendaSerie2015 = new ChartSeries();
    vendaSerie2015.setLabel("Vendas 2015");

    vendas = getVendas();

    for (Venda venda : vendas) {
        vendaSerie2015.set(venda.getLivro().getTitulo(),
                venda.getQuantidade());
    }

    model.addSeries(vendaSerie2015);

    return model;
}
```

Vamos testar, mas as duas séries estão idênticas. Isso porque estamos gerando sempre os mesmos valores por causa do `seed` fixo no método `getVendas`. Para resolver isso, vamos alterar o método `getVendas` para receber o `seed` por parâmetro. Assim, quando chamarmos o método `getVendas`, podemos passar um `seed` diferente, para termos valores diferentes nas séries. Os métodos ficarão assim:

```
public BarChartModel getVendasModel() {

    BarChartModel model = new BarChartModel();

    ChartSeries vendaSerie = new ChartSeries();
    vendaSerie.setLabel("Vendas 2016");

    List<Venda> vendas = getVendas(1234);
```

```
        for (Venda venda : vendas) {
            vendaSerie.set(venda.getLivro().getTitulo(), venda.getQuantidade());
        }

        model.addSeries(vendaSerie);

        ChartSeries vendaSerie2015 = new ChartSeries();
        vendaSerie2015.setLabel("Vendas 2015");

        vendas = getVendas(4321);

        for (Venda venda : vendas) {
            vendaSerie2015.set(venda.getLivro().getTitulo(),
                    venda.getQuantidade());
        }

        model.addSeries(vendaSerie2015);

        return model;
    }

    public List<Venda> getVendas(long seed) {

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

        Random random = new Random(seed);

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

        return vendas;
    }
```

Agora fica para vocês verem as outras opções, o que podemos fazer com os gráficos no Primefaces. Uma coisa legal que podemos fazer é a exibição animada, basta habilitá-la passando o valor `true` para o método `setAnimate` do `model`:

```
    public BarChartModel getVendasModel() {

        BarChartModel model = new BarChartModel();
        model.setAnimate(true);

        // restante do método
    }
```

Há várias outras opções de interatividade, para customizar os valores, etc. Não deixem de olhar o *ShowCase* do Primefaces.

Com isso, chegamos ao final do nosso treinamento, em que conseguimos customizar totalmente a nossa aplicação, utilizando componentes ricos e fáceis de se customizar. Até o próximo treinamento! Alias, essa aplicação ainda não está finalizada! No próximo curso vamos arrumar todas as classes usando CDI:

https://cursos.alura.com.br/course/jsf-cdi (https://cursos.alura.com.br/course/jsf-cdi)

O que aprendemos: - Usar o componente `p:chart` - Preencher o gráfico com o modelo - User series para representar uma continuação de valores - Gerar valores aleatórios através da classe `java.util.Random`