09

Programação procedural não!

Olá! Esse é um exercício que você não precisa enviar resposta, mas deve implementar a alteração sugerida!

Aprendemos que o paradigma da orientação a objetos prega uma forte conexão entre dado e comportamento. Contudo, se olharmos a classe `NegociacoesView` vemos que ela foge um pouco dessa ideia. Vejamos parte do seu template:

```
<!-- aluraframe/client/js/views/NegociacoesView.js -->
<td>
    ${model.negociacoes.reduce((total, n) => total + n.volume, 0.0)}
</td>
```

Nesse ponto, iteramos sobre a lista de negociações do modelo `ListaNegociacoes` aplicando a função `reduce` para calcular o volume total. Esta é uma solução procedural onde temos o dado `ListaNegociacoes` de um lado e o comportamento que calcula o volume total do outro, ou seja, em `NegociacoesView` .

Uma solução mais orientada a objetos é criarmos um getter chamado `volumeTotal` em `ListaNegociacoes` . Com essa alteração, nosso template `NegociacaoView` pode acessar esse getter para obter o volume total das negociações. Dessa forma, onde quer que `ListaNegociacoes` seja utilizada, os dados e o comportamento que calcula o volume total caminharão juntos.

Alterando `aluraframe/client/js/app/models/ListaNegociacoes.js` :

```
class ListaNegociacoes {

    constructor() {

        this._negociacoes = [];
    }

    adiciona(negociacao) {

        this._negociacoes.push(negociacao);
    }

    get negociacoes() {

        return [].concat(this._negociacoes);
    }

    esvazia() {

        this._negociacoes = [];
    }

    // novo método
    get volumeTotal() {
       return this._negociacoes.reduce((total, n) => total + n.volume, 0.0);
    }
}
```

Agora, vamos pedir ao modelo que nos retorne o volume total:

```
<!-- aluraframe/client/js/app/views/NegociacoesView.js -->

<td>
    ${model.volumeTotal}
</td>
```

Perfeito! Sempre que alguém precisar saber o volume total pedirá ao modelo `ListaNegociacoes`, por exemplo, para gerar gráficos ou outros tipos de saída.