03

Herança e reutilização de código

Transcrição

Temos duas Views criadas: `MensagemView` e `NegociacoesView`. Se observarmos, ambas posuem um construtor que recebe um elemento, além de possuir a propriedade `elemento`. As duas têm os métodos `_template` e `update`, que são bem semelhantes. A diferença está na forma em que o método `_template` foi implementado e o seu retorno. E se aumentarmos o número de Views, teremos que ter mais `updates`. Atualmente, o método `update` do `NegociacoesView` está assim:

```
update(model) {

    this._elemento.innerHTML = this._template(model);
}
```

Se nos enganamos e escrevermos `innerHtml`, com as letras de `HTML` em caixa baixa, teremos problemas na execução do código. Para evitarmos a repetição, vamos colocar o que as classes têm em comum, apenas em uma, a nova classe receberá o nome de `View`.

```
class View {

    constructor(elemento) {

          this._elemento = elemento;
    }

    update(model) {

        this._elemento.innerHTML = this._template(model);
    }
}
```

A classe `View` recebeu tudo o que as Views tinham em comum: um `constructor(elemento)` - que guardará internamente um `elemento` - e `update()`. Lembrando que o método `_template` possui algumas diferenças nas classe. Depois, removeremos os métodos `constructor()` e `update()`.

Em seguida, para evitarmos duplicar o código, faremos com que `MensagemView` herde todas as características de `View`. Como em JavaScript trabalhamos com o conceito da orientação a objetos que é herança? Podemos dizer que a `MensagemView` é uma `View`:

```
class MensagemView extends View {

  _template(model) {

      return model.texto ? `<p class="alert alert-info">${model.texto}</p>` : '<p></p>';
  }
}
```

Faremos o mesmo com `NegociacoesView`:

```
class NegociacoesView extends View {

    _template(model) {

      return `
      <table class="table table-hover table-bordered">
          <thead>
              <tr>
                  <th>DATA</th>
                  <th>QUANTIDADE</th>
                  <th>VALOR</th>
                  <th>VOLUME</th>
              </tr>
          </thead>
//...
```

Vamos carregar a `View` no `index.html`.

```
<script src="js/app/models/Negociacao.js"></script>
<script src="js/app/controllers/NegociacaoController.js"></script>
<script src="js/app/helpers/DateHelper.js"></script>
<script src="js/app/models/ListaNegociacoes.js"></script>
<script src="js/app/views/View.js"></script>
<script src="js/app/views/NegociacoesView.js"></script>
<script src="js/app/models/Mensagem.js"></script>
<script src="js/app/views/MensagemView.js"></script>
<script>
    let negociacaoController = new NegociacaoController();
</script>
```

Observe que ao carregarmos os scripts, devemos posicionar a `View` antes das outras Views dependentes. Se a `View` for carregada por última, no navegador veremos uma mensagem de erro que nos dirá: `View is not defined`, porque na definição da classe estamos usando herança nas duas Views.

Se digitarmos a seguinte linha no Console...

```
let v = new NegociacoesView()
```

Tudo funcionará corretamente e o arquivo `NegociacoesView` herdará de `View` o método `update()`. Apesar de termos removido o método, ele está sendo invocado. Mas quando carregamos `NegociacoesView` e `MensagemView`, precisamos ter um `constructor()` que recebe o `elemento`. O construtor chamará o `super()` - fazendo referência ao *super class*, a classe pai. Com as alterações, o `NegociacoesView` ficará assim:

```
class NegociacoesView extends View {

    constructor(elemento) {
        super(elemento);
    }
//...
```

E o `MensagemView` ficará da seguinte maneira:

```
class MensagemView extends View {

    constructor(elemento) {
        super(elemento);
    }
//...
```

Se cadastrarmos uma nova negociação, veremos que está tudo funcionando.

Mas, só encontraremos o método `update` na View. Para que as duas Views pudessem herdar da classe `View`, seria necessário adicionarmos no `NegociacoesView` o `MensagemView` e o `extends`.

Evitamos a duplicação do código, mas será que existe alguma outra falha? Veremos mais adiante.