04

Construtor vs super

Transcrição

Se definimos que "para a criação de uma View, esta deverá herdar de `View` ", pode ocorrer que o desenvolvedor esqueça de implementar o método `_template` . O arquivo `MensagemView` ficaria assim:

```
class MensagemView extends View {

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

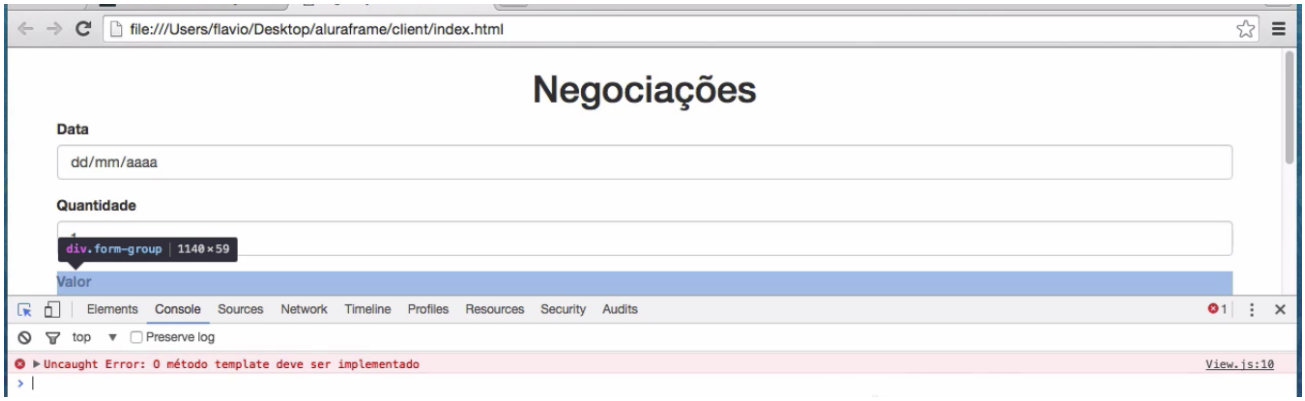
Se executarmos o código no navegador, teremos um erro.

O erro ocorrerá porque o método `update` depende do `_template()` para funcionar. E na classe `View` , não podemos definir a implementação do método, considerando que este sofre variações nas classes filhas, logo, estas serão as responsáveis por definir o `_template()` . Por isso, vamos encontrar uma forma de lembrar ao desenvolvedor que ele deve usar o método `_template()` .

No arquivo `View.js` , vamos adicionar o `_template()` , que lançará um `new Error` .

```
_template() {
    throw new Error('O método template deve ser implementado');
}
```

A mensagem informará que o método template deve ser implementado. Mas se `NegociacoesView` possui um método definido com o nome `_template()` - também utilizado na classe pai - a classe filha irá sobrescrevê-lo. Isto significa que o método válido é o `_template()` de `NegociacoesView` . O mesmo ocorrerá com `MensagemView` . Desta forma, a mensagem de erro só será adicionada caso o desenvolvedor se esqueça de implementar o método nas Views.



Como não adicionamos o `_template()` no `MensagemView.js`, fomos avisados no Console.

Na linguagem JavaScript, **não existem classes abstratas** e, por isso, não podemos obrigar as classes filhas a implementarem o `_template()`. Explicado isto, vamos adicionar novamente o método `_template()` no `MensagemView.js`:

```
class MensagemView extends View {

    constructor(elemento) {
        super(elemento);
    }

    _template(model) {

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

A mensagem de erro não será exibida quando recarregarmos a página no navegador. Para finalizar, faremos um pequeno ajuste. Como foi convencionado, ao usarmos o prefixo `_` no nome do método `_template()`, mesmo as classes filhas não poderiam chamar o método. Apenas a classe pai deveria ter este acesso. Por isso, vamos remover o `_` de todas as referências ao método `_template`.

O trecho referente em `View.js` ficará assim:

```
template(model) {

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

update(model) {
    this._elemento.innerHTML = this.template(model);
}
```

Em `NegociacoesView`, o método `template()` ficará assim:

```
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>
//...
```

E por último, em `MensagemView`:

```
template(model) {

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

Se tivéssemos mantido o prefixo `_`, o código funcionaria corretamente. Mas como se trata de uma indicação de `private`, por convenção, as classes filhas não poderiam sobescrever o método. Com isto, organizamos o nosso código baseados em herança.