03

Métodos estáticos

Transcrição

Precisamos usar uma instância para invocar os métodos, quando queremos usar o `DateHelper`. Mas será que este possui alguma propriedade? Não. Nós configuramos um construtor diferente do padrão para o `DateHelper`? Não. Vale ressaltar que se não definimos um `constructor`, por padrão, é como se existisse vazio.

```
class DateHelper {

  constructor() {}

  dataParaTexto(data) {

      return data.getDate()
          + '/' + (data.getMonth() + 1)
          + '/' + data.getFullYear();

}

textoParaData(texto) {

  return new Date(...texto.split('-').map((item,indice) => item - indice % 2));

}
```

Como não definimos o `constructor`, não colocaremos propriedades na classe. Outra opção seria adicionar a variável `helper2`:

```
adiciona(event) {

    event.preventDefault();

    let helper = new DateHelper();
    let helper2 = new DateHelper();

    let negociacao = new Negociacao(
        helper.textoParaData(this._inputData.value),
        this._inputQuantidade.value,
        this._inputValor.value
    );

    console.log(negociacao);
    console.log(helper.dataParaTexto(negociacao.data));

}
```

Ou seja, adicionaríamos uma instância em todos os lugares que precisasse do `helper`. Isto tem um impacto no uso da memória, apesar de insignificante no nosso caso. Uma melhor decisão é acessar diretamente o método da classe.

```
adiciona(event) {

    event.preventDefault();

    let helper = new DateHelper();
    let helper2 = new DateHelper();

//...
```

Nós substituímos a instância `helper` por `DateHelper`.

```
adiciona(event) {

    event.preventDefault();

    let helper = new DateHelper();
    let helper2 = new DateHelper();

    let negociacao = new Negociacao(
        DateHelper.textoParaData(this._inputData.value),
        this._inputQuantidade.value,
        this._inputValor.value
    );

    console.log(negociacao);
    console.log(DateHelper.dataParaTexto(negociacao.data));
}
```

Nós não queremos que o método seja uma instância do `DateHelper`, queremos poder invocá-lo diretamente da definição da classe. Para isto, no arquivo `DateHelper.js`, adicionaremos o `static` aos métodos:

```
class DateHelper {

  static dataParaTexto(data) {

      return data.getDate()
          + '/' + (data.getMonth() + 1)
          + '/' + data.getFullYear();

}

  static textoParaData(texto) {

      return new Date(...texto.split('-').map((item,indice) => item - indice % 2));
    }
```

Agora, os métodos serão invocamos diretamente da classe e o `NegociacaoController` ficou um pouco mais enxuto. Vamos testar submeter os dados no formulário:

!

Vimos uma novidade em termos de orientação para objeto: a classe `DateHelper` tem métodos estáticos, o que torna desnecessário a criação de uma instância.

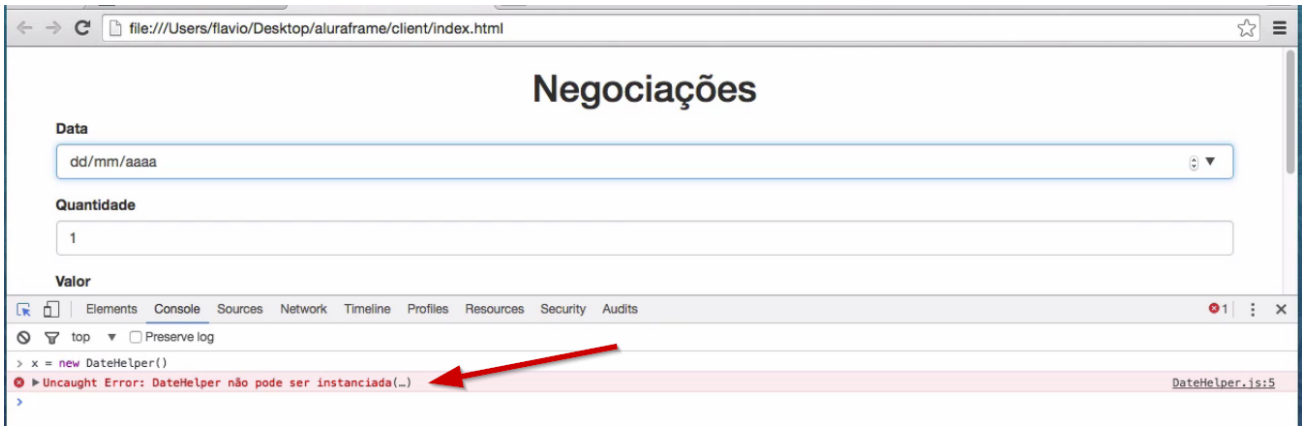
Em seguida, vamos definir um `constructor()`, depois, adicionaremos a função `throw new Error()`. Várias funções do JavaScript não foram migradas para classe e são construtoras.

```
class DateHelper {

    constructor() {

        throw new Error('DateHelper não pode ser instanciada');

//...
```

No Console, digitaremos :

```
x = new DateHelper ()
```

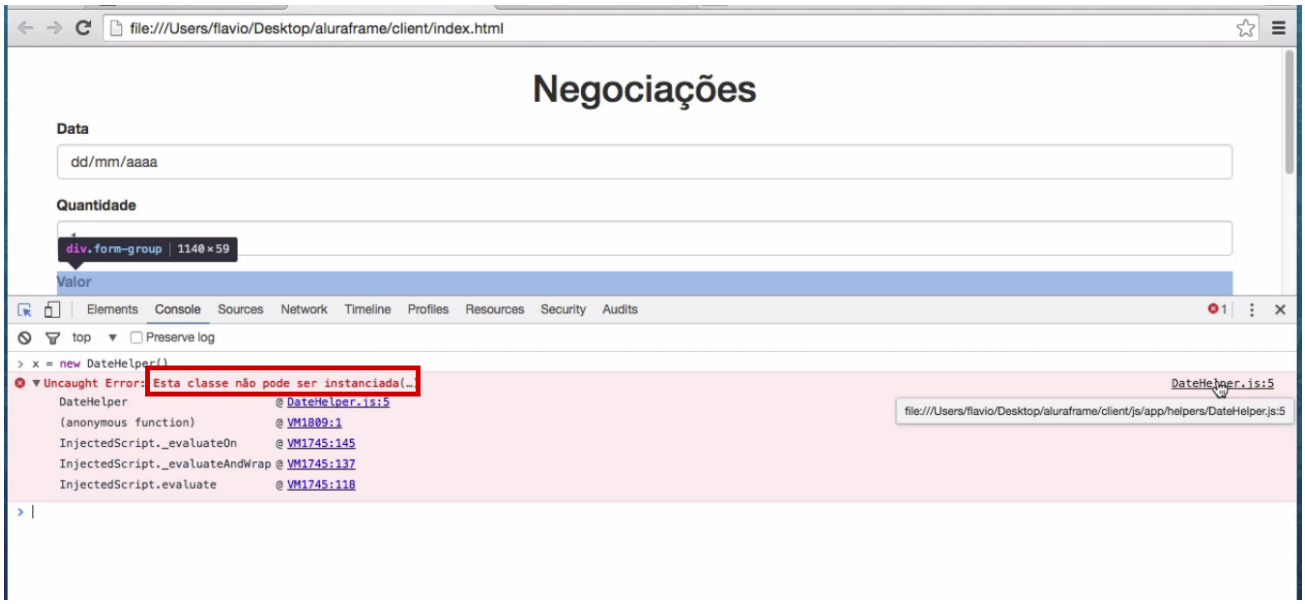
Veremos o seguinte retorno.



Ao ver esta mensagem, o programador saberá que trabalhamos com métodos estáticos. Se clicarmos no erro, veremos qual é a classe e onde está o problema. Faremos um pequeno ajuste na mensagem que aparecerá, deixando-a mais genérica:

```
class DateHelper {

    constructor() {
```

```
            throw new Error('Esta classe não pode ser instanciada');
        }
    //...
```



Mais adiante, vamos melhorar ainda mais a classe.