02

A classe Negociação

Transcrição

A seguir, criaremos a classe `Negociacao`. Nós não usaremos mais a classe `index.js`. Nós também seguiremos a seguinte convenção: dentro da pasta `js`, encontramos a `app`, que contêm todo o código da aplicação. Dentro dela, encontraremos a subpasta `models`, em que estão armazenados todos os modelos. Também convencionaremos que queremos criar a classe `Negociacao.js`, o script que iremos criar, começará em caixa alta (com letra maiúscula). Isto é um pouco incomum, mas isto deixará claro que o JS é uma classe. No arquivo `index.html`, importaremos a `Negociacao.js`.

Para criarmos a classe com o ES6, usaremos a sintaxe `class Negociacao`. Ela terá o mesmo nome do arquivo, propositalmente, para que haja uma paridade e maior organização - mas não era obrigatório. E como definiremos os atributos de uma classe no ES6? Utilizando a função `constructor()`.

```
class Negociacao {

    constructor() {

        this.data = new Date();
        this.quantidade = 1;
        this.valor = 0.0;
    }
}
```

Especificamos que a negociação terá: `data`, `quantidade` e `valor`. Toda negociação terá a data atual por padrão, logo adicionamos o `Date()`. Para `quantidade`, começaremos com o valor padrão `1` e o `valor`, começará de `0.0`.

Em seguida, no arquivo `index.html`, vamos incluir uma nova tag `<script>`. Nosso objetivo será criar duas instância de negociação, ou seja, dois objetos criados a partir da classe `Negociacao`. Usaremos as variáveis `n1` e `n2`.

```
<script>

    var n1 = new Negociacao();
    console.log(n1);

    var n2 = new Negociacao();
    console.log(n2);

</script>
```

Vamos recarregar o navegador e abrir o Console. Veja que já aparecem duas `Negociacoes`:

Ambas ganharam uma `data`, `quantidade` e o `valor`. Observe que nenhuma das `Negociacoes` tinham as propriedades especificadas no código. Elas foram impressas desta forma porque seguem a mesma classe.

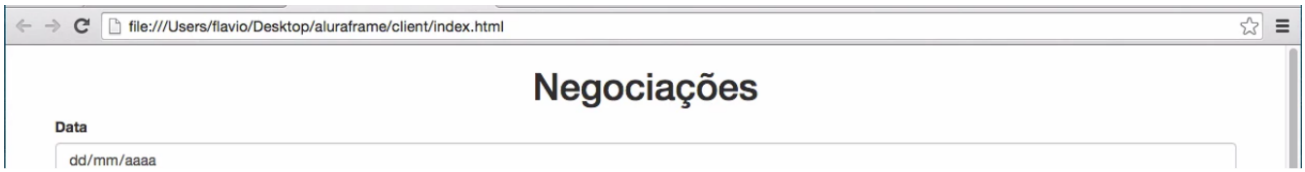
Observe que utilizamos a palavra `new` no nosso código, dentro das variáveis:

```
var n2 = new Negociacao();
console.log(n2);
```

Vejamos o que aconteceria se ela não fosse utilizada na variável `n2`:

```
var n2 = Negociacao();
console.log(n2);
```

No navegador, uma mensagem de erro seria exibida no Console.



Somos informados que a classe `constructor` não pode ser invocada sem operador `new`. O operador `new` é o responsável pela inicialização do `this` de cada objeto criado. Cada objeto terá seu this com suas propriedades. O JavaScript é muito bonzinho e nos avisa do nosso erro:

```
class Negociacao {

    constructor() {

        this.data = new Date();
        this.quantidade = 1;
        this.valor = 0.0;
```

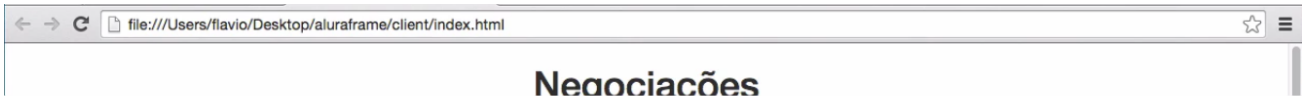
```
    }
}
```

Isto ocorre porque será o operador `new` que fará o `this` ser correspondente ao objeto criado no `index.html`. Então, se especificamos que a `quantidade` da variável `n1` é igual a `10`, saberemos que alteramos o valor unicamente da mesma. A variável `n2` continuará com o mesmo valor.

```
<script>

    var n1 = new Negociacao();
    n1.quantidade = 10;
    console.log(n1);

    var n2 = new Negociacao();
    n2.quantidade = 20;
    console.log(n2);

</script>
```

O operador `new` é bastante importante por ser o responsável em criar um novo `this` para cada instância da classe. O `this` é uma variável **implícita** que sempre apontará para a instância que está executando a operação naquele momento.

Agora, se executarmos o código, no Console do navegador, veremos que cada `Negociacao` terá um valor diferente.



Caso o `this` não variasse para cada instância, quando alterássemos o valor da `quantidade` de `n1`, mudaríamos também a `quantidade` de `n2`. Por isso, o JavaScript não me deixa invocar uma instância sem adicionar termo `new`.

Então, realizamos o primeiro teste com duas estruturas semelhantes, mas com propriedades de valores diferentes.