04

Encapsulamento

Transcrição

Já conseguimos trabalhar com a classe `Negociacao` ,precisamos implementar a seguinte regra de negócio: após criada a negociação, esta não poderá ser alterada. Mas até, agora, podemos fazer alterações.

Vamos ver o que acontece se adicionamos um valor para `quantidade` diferente do que especificamos no parâmetro.

```
<script>

    var n1 = new Negociacao(new Date(), 5,700);
    n1.quantidade = 10;
    console.log(n1.quantidade);
</script>
```

Qual valor será impresso no Console, `5` ou `10` ?

Ele imprimiu o valor recém adicionado `10` . Isto pode causar problemas... Imagine que acabamos de fazer uma negociação e combinamos um determinado valor, mas depois decidimos alterá-lo para benefício próprio. Nosso objetivo é que as propriedades de uma negociação sejam somente para leitura. No entanto, a linguagem JavaScript - até a atual data - não nos permite usar modificadores de acesso. Não podemos dizer que uma propriedade seja apenas leitura (ou gravação). O que podemos é utilizar a convenção de que nos atributos das propriedades de uma classe que não pode ser modificada, usaremos um *underline* (_).

```
class Negociacao {

    constructor(data, quantidade, valor) {

        this._data = data;
        this._quantidade = quantidade;
        this._valor = valor;

    }

    obtemVolume {
```

```
            return this._quantidade * this._valor;
        }
    }
```

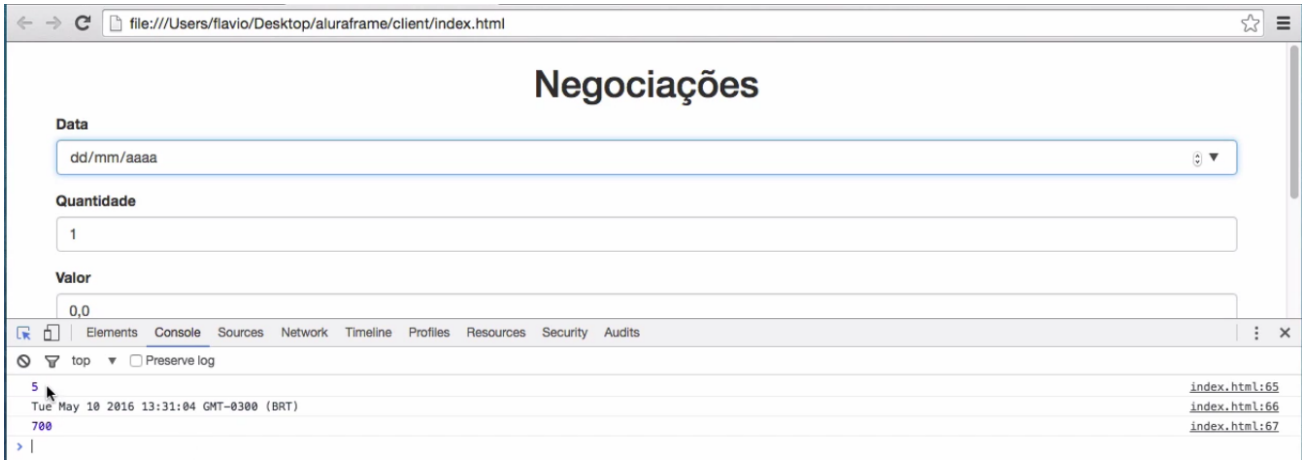
Esta será uma convenção que informará ao programador que as propriedades que contenham `_` só poderão ser acessadas pelos próprios métodos da classe. Isto significa, que mesmo podendo imprimir a propriedade `_quantidade` com outro valor, não deveríamos mais poder acessá-la. O `_` funciona como um aviso dizendo: "programador, você não pode alterar esta propriedade!". Então, se usamos a convenção de utilizar o prefixo, como faremos para imprimir a classe? Se não podemos acessá-la, como podemos fazer isso? Para isto, criou-se métodos chamados acessadores, em que serão utilizados o prefixo `get` . No caso, em `Negociacao.js` , adicionaremos o método `getData()` , que retornará o `_data` . Usaremos também o `getQuantidade()` e o `getValor` que terão finalidades semelhantes.

```
    obtemVolume() {

        return this._quantidade * this._valor;

    }

    getData() {
        return this._data;
    }

    getQuantidade() {
        return this._quantidade;
    }

    getValor() {
        return this._valor;
    }
```

Os métodos da classe poderão acessar os atributos que levam `_` . No entanto, de acordo com a nossa convenção, alguém fora da classe não poderá fazer o mesmo. Por isso, em `index.html` , já que não poderemos chamar `n1._quantidade` , chamaremos `n1.getQuantidade()` .

```
    <script>

        var n1 = new Negociacao(new Date(), 5, 700);
        console.log(n1.getQuantidade());
        console.log(n1.getData());
        console.log(n1.getValor());
    </script>
```

Observe que estamos acessando os demais campos.



Os valores serão impressos corretamente no Console. Em seguida, modificaremos o `obterVolume()` para `getVolume` em `Negocicacao.js` :

```
getVolume() {

    return this._quantidade * this._valor;

}

getData() {
    return this._data;
}

getQuantidade() {
    return this._quantidade;
}

getValor() {
    return this._valor;
}
```

E adicioná-lo no `index.html` :

```
<script>

    var n1 = new Negociacao(new Date(), 5, 700);
    console.log(n1.getQuantidade());
    console.log(n1.getData());
    console.log(n1.getValor());
    console.log(n1.getVolume());
</script>
```

No navegador, veremos que os valores `quantidade` , `data` , `valor` e `volume` serão impressos corretamente.