03

Construtor da classe e métodos

Transcrição

Continuaremos com a construção do código. A primeira ação será apagarmos uma das `Negociacoes`, no arquivo `index.html`:

```
<script src="js/app/models/Negociacao.js"></script>
<script>

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

</script>
```

Agora, definiremos também o `valor` da variável, que será igual a `200.50`.

```
<script>

    var n1 = new Negociacao();
    n1.quantidade = 10;
    n1.valor = 200.50;
    console.log(n1);

</script>
```

O valor já será impresso no Console.

No entanto, existe uma outra forma para passarmos uma instância de `Negociacao`. Imagine que no momento em que criamos uma `Negociacao` e a instância está nascendo, já queremos que seja definida a `quantidade`, `valor` e `data`. Queremos que esses dados sejam passado imediatamente para o construtor.

Se o `contructor()` é uma espécie de função, significa que ela aceitará parâmetros. Considerando isto, adicionaremos os valores para `Negocicao()` no `n1`:

```
<script>

    var n1 = new Negociacao(new Date(), 5, 700);
   console.log(n1);

</script>
```

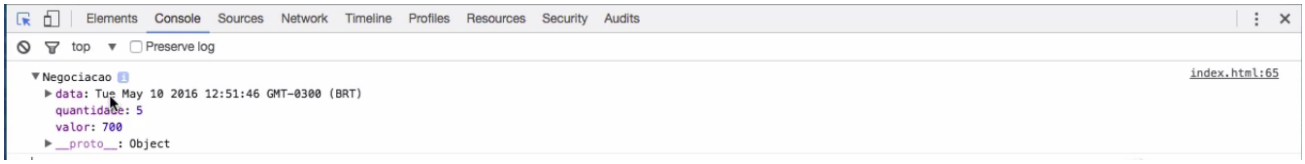
Em `Negociacao.js` , adicionaremos os parâmetros de `constructor()` :

```
class Negociacao {

    constructor(data, quantidade, valor) {

        this.data = data;
        this.quantidade = quantidade;
        this.valor = valor;

    }
}
```

Observe que jogamos os parâmetros para as propriedades.

Atenção: Fique atento sobre a ordem dos parâmetros. Caso a ordem seja alterada, no HTML, os valores das instâncias ficarão incorretos.

Faremos um teste no Console para ver se tudo funcionou corretamente:



Os valores coincidem com os que especificamos no HTML. Porém, ainda não adicionamos o `volume` - a `quantidade` multiplicada pelo `valor` . Faremos isto agora:

```
<script>

    var n1 = new Negociacao(new Date(), 5, 700);
    var volume = n1.quantidade * n1.valor;
    console.log(volume);

</script>
```

Vamos imprimir o `volume` no Console:

O valor `3500` é o resultado da operação de multiplicação. Mas estamos fazendo uma programação procedural, e sempre que o programador precisar do `volume`, ele mesmo terá que realizar este cálculo. Mas então, ele precisaria já saber como calcular o volume? Aumentaria a chance de erro. O que podemos fazer é dotar a classe `Negociacao` com comportamento. A programação orientada a objeto tem uma forte conexão entre o dado e o comportamento. Os dois caminharam juntos.

> Caso você tenha alguma dúvida sobre o assunto, encontrará uma explicação mais detalhada na seguinte exemplificação (https://cursos.alura.com.br/course/javascript-es6-orientacao-a-objetos-parte-1/task/16508).

Em seguida, no `index.html`, em vez de incluirmos o cálculo do `volume` na variável `n1`, faremos uma pergunta para a classe.

```
<script>

    var n1 = new Negociacao(new Date(), 5, 700);
    var volume = n1.obtemVolume();
    console.log(volume);

</script>
```

Depois, teremos que adicionar um método em `Negociacao.js`. Iremos combinar algo: quando criamos uma função dentro de uma classe, nós chamaremos a primeira de **método**. Quando a função estiver fora da classe, continuará sendo chamada de **função**. Então, criaremos o método `obtemVolume` dentro da classe `Negociacao`:

```
class Negociacao {

    constructor(data, quantidade, valor) {

        this.data = data;
        this.quantidade = quantidade;
        this.valor = valor;

    }

    obtemVolume() {

        return this.quantidade * this.valor;
    }
}
```

Agora, ao executarmos o código, o valor `3500` aparecerá novamente no Console:

O valor foi calculado corretamente. O objeto sabe lidar com as propriedades trabalhadas. Logo, a regra de como obter o volume está no próprio objeto. Voltamos a ter uma conexão entre dado e comportamento.