06

Colocando nosso código à prova

Transcrição

Vamos verificar primeiro o cálculo do volume. Vamos lá:

```
// app/js/app.js
let negociacao = new Negociacao(new Date(), 1, 100);
console.log(negociacao.volume); // exibe 100, correto!
```

Como não faz sentido criarmos uma negociação sem as três informações, no ato de criação já passamos esses dados para seu `constructor()`. No entanto, a linguagem JavaScript não me proíbe de fazer assim:

```
// app/js/app.js

let negociacao = new Negociacao();
console.log(negociacao);
```

Outro ponto é que depois de criada, uma negociação não pode ser modificada. Vamos realizar outro teste:

```
// app/js/app.js

let negociacao = new Negociacao(new Date(), 1, 100);
console.log(negociacao);
negociacao.quantidade = 3;
console.log(negociacao.quantidade);
```

Excelente, a atribuição `negociacao.quantidade` é ignorada, porque `quantidade` é um `getter`. No entanto, nada nos impede de acessarmos as propriedade com `_`:

```
// app/js/app.js

let negociacao = new Negociacao(new Date(), 1, 100);
console.log(negociacao);
negociacao._quantidade = 3;
console.log(negociacao.quantidade); // modificou
```

O uso do prefixo é apenas uma convenção para a linguagem JavaScript indicando que a propriedade não deve ser alterada fora da classe, ou seja, uma convenção para indicar que a propriedade é privada.

Por mais que a linguagem JavaScript tenha evoluído, ela não consegue lidar com questões como essa sem termos que apelar para convenções ou aplicar técnicas que aumentam bastante a complexidade do nosso código. Chegou a hora de mostrar um pouquinho do que o TypeScript pode fazer por nós.