14

Para saber mais: Outro critério de igualdade

Agora já sabemos que é boa prática encapsular o critério de igualdade dentro de um método da classe do modelo. Na classe `Negociacao` criamos:

```
class Negociacao {

    //construtor e outros métodos omitidos

    isEquals(outraNegociacao) {
            return JSON.stringify(this) == JSON.stringify(outraNegociacao)
    }
}
```

Assim podemos testar:

```
var hoje = new Date();
var n1 = new Negociacao(hoje, 1, 100);
var n2 = new Negociacao(hoje, 1, 100);

n1.isEquals(n2); //true
```

Agora imagine que o atributo `quantidade` não pudesse ser considerado na hora de comparar duas negociações. Ou seja, uma negociação deve ser igual a outra quando *apenas* a `data` e o `valor` são iguais:

```
var hoje = new Date();
var n1 = new Negociacao(hoje, 1, 100);
var n2 = new Negociacao(hoje, 5, 100);

n1.isEquals(n2); //retorna false, mas queremos true
```

Como encapsulamos o nosso código, já sabemos onde mexer. Mas não podemos mais usar o `JSON.stringify(..)` para testar a igualdade. Lembrando `JSON.stringify(..)` se baseia em todos os atributos! Usaremos cada atributo separadamente:

```
class Negociacao {

    //construtor e outros métodos omitidos

    isEquals(outraNegociacao) {
        return this._data.getTime() == outraNegociacao.data.getTime()
            && this._valor == outraNegociacao.valor;
    }
}
```

Agora podemos testar, mesmo com quantidades diferente os dois objetos são considerados iguais pelo método `isEquals`:

```
var hoje = new Date();
var n1 = new Negociacao(hoje, 1, 100);
var n2 = new Negociacao(hoje, 5, 100);

n1.isEquals(n2); //true
```

Repare na beleza do encapsulamento! Sempre chamaremos o método `isEquals`, indepentemente de qual seja o critério de igualdade concreto.