08

Olhar aguçado para o paradigma da orientação a objetos

Temos o seguinte código que define uma função que sabe validar um código:

```
let codigo = 'GWZ-JJ-12';

function validaCodigo(codigo) {

    if(/\D{3}-\D{2}-\d{2}/.test(codigo)) {
          alert('Código válido!');
      } else {
          alert('Código inválido');
      }

}

validaCodigo('GWZ-JJ-12'); // válido
validaCodigo('1X1-JJ-12'); // inválido
```

Muita coisa acontecendo? Se você não é ninja em expressão regular, vamos desmembrar o código para facilitar sua leitura:

```
function validaCodigo(codigo) {

    // cria a expressão regular. Poderíamos ter usado
    // a sintaxe new RegExp(/\D{3}-\D{2}-\d{2}/)
    // \D é qualquer coisa não dígito
    // \D{3} é qualquer coisa não dígito que forme um grupo de 3 caracteres
    // \d é qualquer dígito.
    let expressao = /\D{3}-\D{2}-\d{2}/;

    // toda expressão regular possui o método test
    // que recebe o alvo do teste, retornando true
    // se passou, e false se falhou
    if(expressao.test(codigo)) {
          alert('Código válido!');
      } else {
          alert('Código inválido');
      }

}

validaCodigo('GWZ-JJ-12'); // válido
validaCodigo('1X1-JJ-12'); // inválido
```

Essa solução é procedural. Veja que toda vez que criarmos um código precisaremos buscar em algum lugar do nosso sistema alguém que o valide. Temos uma separação entre dado e comportamento.

Refaça o código acima adotando o paradigma da orientação a objetos. Uma dica: tudo começa com a criação da classe `Codigo` . Não se preocupe, a ideia aqui é instigar algumas percepções em você sobre este paradigma.