11

# Mãos na massa: extraindo mais códigos

Dando continuidade as boas práticas de organização de código, uma outra que devemos atacar é que nossa função anônima do `form.js` está com muitas responsabilidades, ela sozinha está fazendo muitas coisas. Ela obtém um paciente do formulário, cria a `<tr>` paciente, cria diversos `<td>`, coloca um dentro do outro e depois ainda adiciona-os na tabela!

São muitas funcionalidades para uma única função, vamos quebrá-la em funções menores para melhorar a legibilidade de nosso código:

1- O primeiro passo é extrair a responsabilidade de obter os dados do formulário para uma nova função. Crie a função `obtemPacienteDoFormulario`, que irá cuidar disto. Esta função deve receber o formulário e retornar todos os dados do paciente, e para isto vamos salvar estes dados dentro de um objeto do Javascript :

```
// form.js
function obtemPacienteDoFormulario(form) {

    var paciente = {
        nome: form.nome.value,
        peso: form.peso.value,
        altura: form.altura.value,
        gordura: form.gordura.value,
        imc: calculaImc(form.peso.value, form.altura.value)
    }

    return paciente;
}
```

2- Agora vamos chamar está função no local aonde criávamos várias variáveis com peso, altura,etc..:

```
// form.js
botaoAdicionar.addEventListener("click", function(event) {
    event.preventDefault();

    var form = document.querySelector("#form-adiciona");
    // Remova a criação das variaveis individuais e deixe apenas o objeto paciente
    var paciente = obtemPacienteDoFormulario(form);

    // Restante do código
    ...
});
```

3- A próxima parte que iremos atacar é a criação da `<tr>` paciente. Vamos criar uma função para criar a `<tr>` e uma para as `<td>`. Vamos começar pela função `montaTd`, que deve receber o dado que vai ser colocado dentro da `td` e a classe, e nos retornar o objeto montado:

```
function montaTd(dado, classe) {
    var td = document.createElement("td");
    td.classList.add(classe);
```

```
    td.textContent = dado;

    return td;
}
```

4- Agora aproveitando a função `montaTd` , vamos criar a `montaTr` , que vai receber um objeto paciente, criar cada uma das `td` , e colocar dentro da `tr` :

```
function montaTr(paciente) {
    //Cria TR
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");
    //Cria as TD's e a adiciona dentro da TR
    pacienteTr.appendChild(montaTd(paciente.nome, "info-nome"));
    pacienteTr.appendChild(montaTd(paciente.peso, "info-peso"));
    pacienteTr.appendChild(montaTd(paciente.altura, "info-altura"));
    pacienteTr.appendChild(montaTd(paciente.gordura, "info-gordura"));
    pacienteTr.appendChild(montaTd(paciente.imc, "info-imc"));
    // retorna a TR
    return pacienteTr;
}
```

5- Agora vamos fazer as substituições no event listener:

```
var botaoAdicionar = document.querySelector("#adicionar-paciente");
botaoAdicionar.addEventListener("click", function(event) {
    event.preventDefault();

    var form = document.querySelector("#form-adiciona");

    var paciente = obtemPacienteDoFormulario(form);

    var pacienteTr = montaTr(paciente);

    var tabela = document.querySelector("#tabela-pacientes");

    tabela.appendChild(pacienteTr);

});
```

6- O código ficou bem mais limpo e organizado! Por último , vamos fazer que o form seja limpo após adicionar um paciente. Utilize a função `.reset()` como último comando:

```
var botaoAdicionar = document.querySelector("#adicionar-paciente");
botaoAdicionar.addEventListener("click", function(event) {
    // restante do código
    tabela.appendChild(pacienteTr);

    form.reset();

});
```

Veja que agora temos um código muito mais organizado e fácil de ler!