06

Melhorando ainda mais o código

Transcrição

Outra responsabilidade a ser extraída é a criação da `tr` e das `td` s do paciente. Atualmente o trecho do código do `form.js` está assim:

```
//cria a tr e a td do paciente
var pacienteTr = document.createElement("tr");

var nomeTd = document.createElement("td");
var pesoTd = document.createElement("td");
var alturaTd = document.createElement("td");
var gorduraTd = document.createElement("td");
var imcTd = document.createElement("td");

nomeTd.textContent = nome;
pesoTd.textContent = peso;
alturaTd.textContent = altura;
gorduraTd.textContent = gordura;
imcTd.textContent = calculaImc(peso,altura);

pacienteTr.appendChild(nomeTd);
pacienteTr.appendChild(pesoTd);
pacienteTr.appendChild(alturaTd);
pacienteTr.appendChild(gorduraTd);
pacienteTr.appendChild(imcTd);
```

Na parte de baixo do arquivo, criaremos a função `montaTr` , que receberá um paciente como parâmetro e, como o próprio nome indica, montará a `tr` com os dados:

```
function montaTr(paciente) {
    var pacienteTr = document.createElement("tr");

    return pacienteTr;
}
```

Em seguida, devemos preenchê-la com as `td` s do paciente. Se movermos o código referente aos dados do paciente para dentro da função, ela ficaria mais legível:

```
function montaTr(paciente) {
    var pacienteTr = document.createElement("tr");

    var nomeTd = document.createElement("td");
    var pesoTd = document.createElement("td");
    var alturaTd = document.createElement("td");
    var gorduraTd = document.createElement("td");
    var imcTd = document.createElement("td");

    nomeTd.textContent = paciente.nome;
```

```
    pesoTd.textContent = paciente.peso;
    alturaTd.textContent = paciente.altura;
    gorduraTd.textContent = paciente.gordura;
    imcTd.textContent = paciente.imc;

    pacienteTr.appendChild(nomeTd);
    pacienteTr.appendChild(pesoTd);
    pacienteTr.appendChild(alturaTd);
    pacienteTr.appendChild(gorduraTd);
    pacienteTr.appendChild(imcTd);

    return pacienteTr;
}
```

As tags `td` serão criadas, e então preenchidas com `paciente.nome` , `paciente.peso` , `paciente.altura` , `paciente.gordura` e já não precisaremos calcular o IMC, pois o cálculo foi feito em `paciente.imc` .

Por fim, chamaremos a função `montaTr` quando o botão for clicado. A função ficará dentro da variável `pacienteTr` :

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

    var form = document.querySelector("#form-adiciona");
    // Extraindo informacoes do paciente do form
    var paciente = obtemPacienteDoFormulario(form);
    // Cria a tr e a td do paciente
    var pacienteTr = montaTr(paciente);

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

    tabela.appendChild(pacienteTr);

});
```

Ao preenchermos os dados do formulário, veremos que eles continuarão sendo adicionados à tabela com o código mais legível.

Adicionando classes aos elementos

Ao inspecionarmos o HTML da nossa tabela, veremos que os pacientes adicionados por meio do formulário não possuem algumas características dos pacientes nativos. A `tr` do paciente nativo possui a classe `paciente` , assim como as `td` s - cada uma com uma classe indicando a informação contida na `td` . Já os pacientes que adicionamos com o formulário não possuem classes - tanto na `tr` quanto nas `td` s. Ou seja, não estamos criando um paciente exatamente igual ao paciente nativo.

Vamos alterar o código da função `montaTr` para criar um paciente com as classes corretas. Já sabemos como adicionar uma classe a um elemento, a seguir, adicionaremos a classe `paciente` na `tr` . Para isso, usaremos o método `add()` :

```
function montaTr(paciente){
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");
```

```
    var nomeTd = document.createElement("td");
    var pesoTd = document.createElement("td");
    var alturaTd = document.createElement("td");
    var gorduraTd = document.createElement("td");
    var imcTd = document.createElement("td");

    nomeTd.textContent = paciente.nome;
    pesoTd.textContent = paciente.peso;
    alturaTd.textContent = paciente.altura;
    gorduraTd.textContent = paciente.gordura;
    imcTd.textContent = paciente.imc;

    pacienteTr.appendChild(nomeTd);
    pacienteTr.appendChild(pesoTd);
    pacienteTr.appendChild(alturaTd);
    pacienteTr.appendChild(gorduraTd);
    pacienteTr.appendChild(imcTd);

    return pacienteTr;
}
```

Se testarmos no navegador, veremos que a `tr` será criada com a classe. Porém, falta adicionarmos as classes nas `td` s, por exemplo, `info-nome` e `info-peso` :

```
function montaTr(paciente){
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");

    var nomeTd = document.createElement("td");
    nomeTd.classList.add("info-nome");
    nomeTd.textContent = paciente.nome;
```

Temos que fazer esse código para todas as `td` s, criar o elemento, incluir a classe e o valor. Observe que por termos separado as funções, já sabemos onde fazer as alterações.

Função para criar e montar uma `td`

Quando identificamos códigos repetidos, temos a opção de exportá-los para uma função, que será responsável por eles. A função `montaTd` criará a `td` e adicionará a classe juntamente com o dado. Como a classe e o dado variam de acordo com a `td` , iremos recebê-los por parâmetro na função:

```
function montaTd(dado, classe) {
    var td = document.createElement("td");
    td.classList.add("info-nome");
    td.textContent = paciente.nome;

    return td;
}
```

Agora basta chamar essa função em `montaTr` :

```
function montaTd(dado, classe) {
    var td = document.createElement("td");
    td.classList.add("info-nome");
    td.textContent = paciente.nome;

    var nomeTd = document.createElement("td");
    nomeTd.classList.add("info-nome");
    nomeTd.textContent = paciente.nome;
//...
```

Porém, teríamos que ter diversas linhas de código fazendo a mesma tarefa. O código repetidamente criaria a `td`, adicionaria uma classe e depois o dado. Será que existe alguma forma de simplificarmos esse trabalho? Sim, podemos criar diretamente uma função que monta as tags `td`: `montaTd()`.

```
function montaTd(dado){
    var td = document.createElement("td");
    td.textContent
}
```

No `td` criado, deve ser adicionado como o conteúdo de texto o dado, além de uma classe. Com as alterações, o trecho ficará da seguinte maneira:

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

    return td;
}
```

Depois, chamaremos a função `montaTd()` na variável `pesoTd`.

```
function montaTr(paciente) {
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");

    var nomeTd = document.createElement("td");
    nomeTd.classList.add("info-nome");
    nomeTd.textContent = paciente.nome;

    var pesoTd = montaTd(paciente.peso, "info-peso");

    pacienteTr.appendChild(nomeTd);
    pacienteTr.appendChild(pesoTd)
    pacienteTr.appendChild(alturaTd)
    pacienteTr.appendChild(gorduraTd)
    pacienteTr.appendChild(imcTd)
```

Assim, deixaremos bem claro quais são as responsabilidades do nosso código, separando em uma função a criação da tag `td`. Para cada uma delas, chamaremos a função `montaTd()`.

```
function montaTr(paciente) {
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");

    var nomeTd = montaTd(paciente.nome, "info-nome");
    var pesoTd = montaTd(paciente.peso, "info-peso");
    var alturaTd = montaTd(paciente.altura, "info-peso");
    var gorduraTd = montaTd(paciente.gordura, "info-gordura");
    var imcTd = montaTd(paciente.imc, "info-imc");

    pacienteTr.appendChild(nomeTd);
    pacienteTr.appendChild(pesoTd)
    pacienteTr.appendChild(alturaTd)
    pacienteTr.appendChild(gorduraTd)
    pacienteTr.appendChild(imcTd)

    return pacienteTr;
}
```

É possível adicionarmos o paciente pelo formulário e inspecionarmos o seu HTML. Assim, sua estrutura, incluindo as classes, está igual à de um paciente nativo.

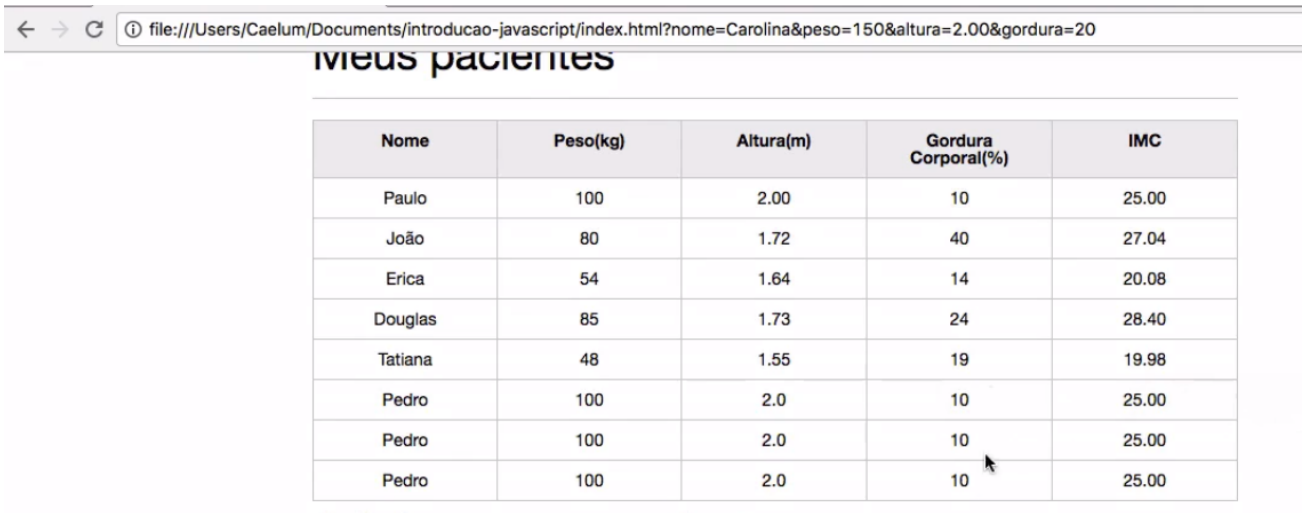
Limpando o formulário após adicionar o paciente

É possível "enxugar" ainda mais o nosso código, adicionando o `montaTd` diretamente no `appendChild()`.

```
function montaTr(paciente){
    var pacienteTr = document.createElement("tr");
    pacienteTr.classList.add("paciente");

    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"));
```

Após adicionarmos um paciente na tabela, os dados continuarão no formulário.



É recomendável limpá-los para não corrermos o risco de adicionarmos pacientes iguais. Poderemos limpar os campos do formulário chamando a função `reset()` depois de inserirmos o paciente na tabela.

```
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);


    form.reset();
});
```

Desta forma, quando um paciente é adicionado, os dados no formulário serão apagados.

Nesta aula vimos boas práticas, aprendemos que não é bom trabalharmos com um código gigantesco em um único arquivo - o ideal é quebrá-lo, dividindo as responsabilidades em diferentes arquivos, simplificando a manutenção. Se temos um problema no formulário, por exemplo, saberemos que devemos trabalhar com o arquivo `form.js` - em vez de buscarmos o erro em um arquivo de 300 linhas. Além disso, vimos que também é boa prática separarmos as responsabilidades em funções, deixando o código mais legível. Cada linha tem uma função específica, independente ao restante do código, que está mais organizado e legível.

Falamos também sobre os objetos do JavaScript, que possuem características e representam coisas do mundo real. No nosso caso, o objeto representa um paciente, mas já havíamos trabalhado com outros, porém, sem denominá-los como tal. Para criarmos um objeto, usamos as chaves (`{}`) e as propriedades, separando com vírgula as diferentes características.

Esta aula foi sobre boas práticas e refatoração, para que você possa se tornar um excelente desenvolvedor. Continuaremos a seguir!