01

Jasmine Aula 2

Transcrição

[00:00] Bem vindo à segunda aula do curso de testes em java script com o Jasmine. Na primeira aula você aprendeu a testar a classe "MaiorEMenor", escrever um monte de testes para ela. Nessa segunda aula a ideia é continuar a escrever testes, até para você praticar isso e a ideia de testes automatizados ficar natural. Agora, em um outro problema, claramente, e começar a discutir um pouquinho de como escrever bons testes e pensar em cenários, etc.

[00:33] Eu vou aqui pro meu sublime, que eu tinha o código exatamente de onde eu parei, só que agora eu vou começar uma nova classe. Eu vou criar um arquivo aqui que eu vou chamar de "Paciente.js". Imagine que estamos fazendo um sistema médico, onde eu tenho pacientes, e esses pacientes eles passam por consultas e tudo mais.

[00:57] Vou começar aqui com function "Paciente", e a função vai receber um nome, uma idade, um peso e uma altura. Eu vou aqui programar também pensando um pouquinho minha orientação a objetos e tudo mais. Var clazz e ele já retorna classe. O primeiro método que meu paciente vai ter é o que vou chamar de imprime. Um método bem simples, ele vai só mostrar um alert para mim na tela. Então, alert, o nome da pessoa, a idade em anos. Simples. Vamos para o próximo.

[01:32] O próximo é mais legal. Vou chamar esse método de batimentos. Aqui eu vou calcular a quantidade de batimentos que o coração dele já deu na vida. Como eu vou saber isso? Bem, em média um coração bate por volta de 80 vezes por minuto. Então, vou pegar a idade dele, vou multiplicar por 365 dias em ano e vou multiplicar por 24 horas, que é o que eu tenho em um dia, vou multiplicar por 60 minutos, que é o que eu tenho em cada hora, e vou multiplicar por 80. Isso aqui é uma aproximação do batimento, da quantidade de vezes que o coração da pessoa bateu.

[02:07] E a próxima função que é importante para a nós, eu vou chamar de IMC. IMC é aquele índice de massa corpórea, aquela conta que é feita com o teu peso e com a tua altura, que te disse se você está mais ou menos saudável. Então, a forma de IMC, você pode até olhar na internet é o peso dividido pelo quadrado da altura. Obviamente, os médicos pegam esse número e te falam se você está saudável. Eu tenho essa classe "Paciente", vamos começar a testar ela.

[02:40] Pra testar ela vamos lá na pasta spec, já aprendemos isso. Então, todo o código está no source. Eu venho no spec, spec na pasta spec, por questão de organização. Então, todo spec começa com um describe. Eu vou descrever um paciente, vou passar uma função anônima, e vou escrever o primeiro teste. It "deve calcular o IMC", function. A primeira coisa é criar a classe "Paciente" que é a classe que eu quero testar. Vou fazer new paciente, ou paciente Guilherme, que tem 28 anos, pesa 72 quilos e tem 1.82 de altura.

[03:25] O método que eu quero testar é o método IMC, "guilherme.imc", e eu vou colocar aqui "var imc" da pessoa. Eu vou fazer aqui "expect(imc).toEqual", e o IMC dele tem que ser 72 dividido 1.82, vezes 1.82. É o IMC que eu espero dele. Escrevi o teste, tenho o código de produção, vou rodar. Dá uma olhada, eu rodei e não mudou nada, só rodou o "MaiorEMenor". Por quê? Porque eu não posso esquecer de ir no "SpecRunner" importar a classe.

[03:58] Faltou incluir o paciente, e faltou incluir o "PacienteSpec". Rodou, o teste está verde, o IMC deu certo. Agora, a pergunta que eu tenho para você é o seguinte: esse código funcionou? Para o paciente Guilherme, que tem 28 anos, 72 quilos e 1.82. Será que esse código funciona para outros pesos e alturas? Como garantir isso? Só escrevendo um teste. Veja só o que eu vou fazer, eu vou colocar aqui o "deve calcular o IMC 2". Eu vou colocar outra altura qualquer aqui, então, serão 82 quilos e 1.77. E mudar a fórmula também, 82, 1.77 por 1.77. Salvei e rodei.

[04:49] O "deve calcular IMC" e "deve calcular IMC 2" passaram, eu estava esperando isso. Só que a pergunta é: eu testei agora para 72 quilos, 1.82 de altura, 82 quilos, 1.77 de altura, e para os outros? O problema é: eu vou entrar nesse

looping infinito de testar todos os possíveis cenários e não dá, a quantidade de combinações é gingante, eu não consigo. O que eu faço? Nesse caso, usamos uma coisa na área de testes que nós chamamos de classes de equivalência, o que é uma classe de equivalência? Uma classe de equivalência é um conjunto de cenários onde todos eles exercitam o código da mesma maneira.

[05:28] Para o nosso código paciente, na função IMC em particular, não importa o peso, não importa a altura, o cálculo é sempre o mesmo. Então, o ponto é: o cenário 72 quilos e 1.82, do ponto de vista do teste é idêntico a 82 e 1.77, e a prova disso é que se por algum motivo tiver um bug na fórmula, um vezes 2 aqui sobrando, os dois testes falham. Então, de nada adianta eu ter dois testes que sempre passam juntos e sempre falham juntos.

[06:02] Isso mostra que estamos de certa forma testando o mesmo cenário. Nesses casos, o que que eu faço? A boa prática é um único teste por classes de equivalência. Isso vai facilitar sua manutenção e, obviamente, o seu desafio é encontrar as várias classes de equivalência diferentes. Que são, de novo, cenários que exercitam partes diferentes do seu código. Quando tínhamos lá no maior e menor, com nosso loop tínhamos o cenário de um único número na array, porque meu código pode funcionar diferente, n números, em ordem crescente, decrescente e tudo mais.

[06:42] O teu exercício é pensar nessas várias classes de equivalência, é isso que faz a diferença. Mas percebeu que tem dois testes iguais? Joga fora e tenha um só. Aproveitando aqui, outra coisa que eu vou mostrar para vocês é que eu tinha “var imc” e colocado variável, mas eu posso fazer inline, muitos desenvolvedores preferem fazer inline, tudo continua funcionando do mesmo jeito. Agora, a próxima classe que eu vou fazer é a classe consulta.

[07:11] Vou na minha pastinha source, vou criar aqui o “consulta.js”. E uma consulta, “function Consulta”, vai ser com um paciente e uma lista de procedimentos que ele fez naquela consulta. Se a consulta foi particular ou se ela foi pelo convênio, ou se ela é um retorno de uma consulta que foi realizada no passado. Var clazz, já estamos acostumados com isso. O método que eu vou dar aqui é um método que chama “preco”. Esse é o cara que vai calcular o preço da minha consulta.

[07:46] Como vai funcionar? A regra é a seguinte, se for retorno não paga nada, o valor da consulta é zero. Se for particular, ele vai multiplicar o valor da consulta, mas como eu cálculo o valor consulta? De acordo com o procedimento, eu tenho um valor diferente, e eu posso ter vários procedimentos. A minha implementação pra isso vai ser o seguinte, se for retorno já vou retornar zero desde o começo. Vou criar uma variável “precoFinal”, e vou fazer um loop nessa lista de procedimentos. Então, para cada procedimento na lista a regra vai ser o seguinte.

[08:23] Se o procedimento for raio-x, então, eu pego o preço final e somo 55, que é quanto custam um raio-x, else if. Se o procedimento for um gesso, eu vou somar 32 reais. Caso contrário, o preço do procedimento é padrão e é 25. Acabou loop. Se for particular, eu dobro o valor, preço final vezes igual a dois, e eu retorno o preço final. E a nossa implementação aqui no cálculo de preço. Veja só que eu tenho uma série de parâmetros diferentes e dá para exercitar esse código de maneiras diferentes.

[09:13] Então, eu vou ter vários testes, por exemplo, um pra retorno, um para particular, um pra procedimentos genéricos, um pra procedimento pra raio-x, outro pra gesso, etc. Vamos para o teste. O primeiro teste que eu vou fazer, vou lá na pasta spec, então, criar esse cara. Vou salvar como “ConsultaSpec.ks”. Describe consulta. Vamos escrever essa infra que eu preciso. Function. O primeiro cenário que eu vou testar é o retorno, porque o cenário é mais fácil. Então, "não deve cobrar nada se for um retorno".

[10:01] Vou criar um paciente, porque eu preciso de um paciente.. Meu paciente Guilherme, 28 anos, 72, 1.80 de altura, e eu vou criar uma consulta. New consulta, Guilherme, não vou passar nenhum procedimento, que eu não preciso. É particular. É um retorno. Eu sei que eu espero que a “consulta.preco” seja igual a zero. SpecRunner de novo, adiciono as classes.

[10:43] Então, “Consulta”, “ConsultaSpec”. No browser, rodei, tudo verde meu teste passa. Vamos só ver o teste falhar, é sempre uma boa prática, se eu colocar return 1 o teste falha. Coloca zero de novo. Teste funciona quando implementação está certa e falha quando a implementação está errada, eu preciso saber isso. Então, agora vamos para o próximo cenário, eu vou garantir que um procedimento comum é cobrado 25 reais.

[11:17] It, deve cobrar 25 por cada procedimento comum. Function. Vou copar essas duas linhas aqui que elas são parecidas. A diferença é que vou mudar os parâmetros, vou passar dois procedimento. Proc1 e proc2, o nome do procedimento não importa muito. Aqui não é particular e também não é um retorno, se não ele não vai conseguir calcular. Eu espero que o preço da consulta seja 50. Rodando, tudo verde, tudo funcionando, está perfeito.

[11:56] Agora o que é uma discussão interessante, veja o que eu passei dois parâmetros, dois procedimentos, eu podia ter passado 3, 4, 5, 6, 20, 30, 40. Mas qual a diferença de 2 para 3 procedimentos e para 4 procedimentos? Para esse código, em particular, nenhuma. Então, eu penso, dado que eles são da mesma classe de equivalência em um cenário mais simples. Se eu quiser fazer com 3, não tem muita diferença. Se colocar aqui 75 o teste vai passar.. Um, dois, três é fácil, é pequeno e tudo mais.

[112:28] Mas não vou fazer um teste com 30 procedimentos sendo que o teste com três é a mesma coisa. Então, começa a pensar nisso também, na classe de equivalência faz o mais simples, o mais claro de ser lido. Agora, vamos lá para o próximo teste, vou chamar de “deve cobrar preço específico dependendo do procedimento”, function. Vou criar o Guilherme do mesmo jeito. Ctrl+c e Ctrl+v aqui, vou criar consulta, passando o Guilherme.

[13:07] A lista de procedimento vai ser assim, procedimento comum, raio-X, outro procedimento comum, gesso. False, false. O que eu espero? Espero que o preço seja igual a 25, que é o procedimento comum. 55 é o raio-x, 25 é outro comum, 32 que é o gesso. Vamos ver? Se eu rodar, tudo continua passando. Já escrevi três testes com vocês. Não cobra nada se for retorno, 25 reais cada procedimento, deve cobrar um preço específico.

[13:52] Veja só que eu tenho mais cenários, eu não testei o caso do particular, eventualmente o caso do particular com um procedimento especial, raio-x, gesso, etc. O que eu quero que saia dessa aula para vocês é a parte de classes de equivalência. Essa é a parte importante. Então, começou a escrever teste, pensa nas diferentes classes de equivalência. Se pensou dois cenários que pertencem à mesma classe, escreve um único teste.

[14:17] E tenta optar pelo mais simples, pela menor classe de equivalência possível. Aqui eu poderia passar dois procedimentos. Eu não quero que você passe quinze, porque não vai fazer diferença, só vai dar trabalho. Então, no exercício você vai continuar escrevendo teste para essa classe consulta, e continua pensando nos outros cenários.

[14:40] Um detalhe que eu não falei, você deve ter reparado naturalmente, que a palavra consulta que você escreve aqui nunca aparece aqui, é pra isso que serve o describe. Se eu colocar à consulta X, ele vai exibir consulta X. Foi um exemplo bobo, mas mostra pra você que o texto que eu escrevo ali é o que aparece aqui. Então, a segunda aula é isso, classe de equivalência, continuar testando, pensar em cenários diferentes para exercitar o seu código. Obrigado por esse capítulo e espero você no próximo.