03

Resolvendo bugs

Transcrição

[00:00] Repare agora que a nossa função ainda tem um laço. É claro, quanto mais complexo um código mais difícil de entender, maiores as chances de um bug. Será que conseguimos tirar o laço para ficar mais fácil entender o que a função como unidade faz?

[00:25] Se eu tirar o for, ele limpa só uma casa para a direita. Faz sentido. Eu queria limpar uma para a direita e a próxima para a direita. Poderia dar um copypaste. Se o que estou fazendo é chamar o mesmo código, posso dizer para ele remover do mapa a posição com determinada quantidade. Quando eu chamo eu mesmo, vou andar mais uma para a direita. Vira um ciclo e vou limpando cada casinha da direita.

[01:55] Só que eu não parei. Eu falei e não acabei. Chamei o remove, que chamou o remove, que chamou o remove. Parece que não tem fim. Precisamos parar, senão empilhamos uma chamada de método em cima da outra. Isso vai ocupando espaço na memória. Uma hora a memória ou o limite da pilha vai explodir. Foi exatamente o que ele disse. Um looping infinito de invocação começa a ocupar o espaço em cima do outro e estoura. Um looping infinito do tipo for fica rodando feito louco. Um looping infinito desse tipo quebra a aplicação. Os dois trazem problemas. Não queremos isso.

[03:33] Queremos parar quando não temos mais nada para andar para a direita. Ou seja, quando andamos zero. Eu vou chamar o remove com três, depois com dois, depois com um, e paro.

[04:26] É claro que o fluxo vai crescer mesmo se a quantidade for menos um, porque nunca parei. Ele dá o mesmo erro. Cada um dos puts é uma chamada do próprio método chamando ele mesmo. Estamos empilhando as chamadas e a pilha fica tão grande que uma hora estoura.

[05:23] Queremos dar um critério de parada. Chamar você mesmo é a recursão. É o que estamos fazendo agora. Só que removemos sem parar. Se não paramos, uma hora estoura. Precisamos mandar ele parar.