11

Posição

Transcrição

[00:00] Legal, escrevemos, por exemplo, a função posição válida. Ela recebe um mapa e pega a posição que tem ali dentro, que é uma linha, uma string. O nosso mapa é uma array de strings. Mas repare que a qualquer instante eu acesso uma posição no mapa. Pego, por exemplo, o terceiro elemento de uma array e o segundo elemento. Ele se comporta quase como um array, porque tem como acessar o cara que é chamado de colchetes. Não só isso, como também permite colocar outras coisas nele, imprimir o nome, o size.

[01:22] Se parece tanto um array, ter como concatenar elementos, descobrir o tamanho, não estamos nem aí se é um array ou não. Estamos preocupados que ele disponibilize para nós uma maneira de acessar uma posição aleatória, uma maneira de adicionar elementos no final. Uma maneira de calcular o número de elementos que tenho lá dentro. Independe de ser um array, ou uma string, ou qualquer outra coisa que tenha esses comportamentos, se ele tem esses comportamentos estou feliz, porque ele faz o que eu queria que fizesse.

[02:15] Em programação, chamamos de duck typing. Não estou preocupado com o tipo com que estou trabalhando. Não estou preocupado se o mapa é uma array de caracteres. Ou uma string de strings. Ou um mapa. Não estou preocupado com o tipo, mas sim se ele se comporta como. Se ele se comporta como uma array, estou feliz. Isso fica muito famoso e é chamado de duck typing.

[02:57] Duck em inglês é pato. Se ele nada como pato, grasna como pato, e anda como pato, para mim é suficiente para tratar ele como pato. Quando eu mandar ele fazer algo, imagino que vá fazer como se fosse um pato. Não importa se é ou não.

[03:30] Essa é uma sacada bem interessante do Ruby e de outras linguagens que implementam o duck typing, porque não ficamos atrelados a verificar o tipo para poder aí então invocar o grasna, o anda, o nada, e saber que ele faz isso da maneira que queremos. No Ruby, basta chamar. Se ele tem esse comportamento, beleza. Se não, vai dar erro. Não estou preocupado com o tipo específico, mas sim se ele tem o comportamento ou não.

[04:05] Só que como toda característica, temos uma vantagem e desvantagem. Qual o cuidado que temos que tomar? Imagine que te dou um outro produto que também tem o comportamento size. O que o size de um produto diz? Ele diz o tamanho do produto. Ele não diz quantos elementos tem lá dentro. A verdade é que nem sempre tudo que tem um comportamento que damos aquele nome é o que imaginamos. Nem tudo que grasna, anda e nada é um pato. Ou mais específico ainda, nem tudo que é um pássaro, nada, anda e grasna é um pato. E talvez estejamos assumindo que é, quando executamos não é, e aí até executa, mas causa algo nada a ver. Afinal ele não é um pato. É algo totalmente diferente. E eu achei que ele era um pato.

[05:23] Por não ter uma fase de compilação que verifica a tipagem, temos essa vantagem do duck typing, de não precisarmos nos preocupar com a tipagem, mas ao mesmo tempo corremos o risco dele nem ter o comportamento e de ele ter esse comportamento e ser uma coisa totalmente diferente do que o programador imaginava.

[05:52] Por enquanto vimos que se tem size e tem acesso aleatório, para nós está ótimo, acessamos o mapa felizes e contentes. O duck typing para nós aqui está sendo muito legal, nem precisamos nos preocupar com o tipo até agora.