01

Composição, herança e os cuidados com mixins

Transcrição

[00:00] Dando uma olhada na situação da nossa classe “livro”, reparamos que o construtor ainda faz bastante coisa. Ele atribui diversas variáveis membro. E é claro, esse código está muito parecido em todos os produtos.

[00:18] Vamos tentar pegar ele, e colocar no nosso produto. Vamos no produto, definimos o construtor, o título, é comum a todos eles: o preço, ano de lançamento. Reimpressão não, editora sim, e sobrecapa não.

[00:32] Então esses campos eu mantenho, e os outros dois campos, eu vou manter só no livro. Certo? Só reimpressão e sobrecapa no livro. Agora eu quero na hora que inicialize o meu produto, passar o título, preço, lançamento e editora.

[00:50] Mas o “produto.initialize” eu estou inicializando eu mesmo. Então eu vou chamar o construtor do módulo que eu acabei de incluir, o “super”. Porque é o construtor que estava logo acima de mim.

[01:06] (título, preço, ano_lançamento e editora). Os outros dois campos, eu atribuo como variável membro. Vamos executar o código, e está tudo certo. Perfeito.

[01:24] Reparem que na revista, vamos poder fazer a mesma coisa: remove os quatro campos, mantêm só os campos novos, e no ebook também. Remove os outros campos e mantêm só os campos novos. Que no caso, não tem nenhum.

[01:42] Sempre lembrando de chamar o “super”. Rodo tudo de novo. Perfeito. No caso do ebook, como o construtor é idêntico, eu simplesmente removo ele e deixo o construtor que o produto tinha colocado dentro da minha classe. Executo e está tudo certo.

[02:01] Vamos dar uma olhada em algum outro tipo de comportamento comum entre os produtos. Por exemplo, “possui_reimpressao”. Todo o produto que é um impresso, possui reimpressão.

[02:10] No meu caso, eu vou extrair um comportamento, um módulo, chamado “include Impresso”, que define o método “possui_reimpressao”. Vou fazer o require do impresso, criar o arquivo “impresso.rb”, definir o módulo “impresso”, e colocar esse método lá dentro.

[02:32] Vou definir o construtor do impresso, recebendo “possui_reimpressao”, e vou atribuir a variável membro. Agora a dúvida: quando eu chamar esse método “super”, quem eu vou chamar? O “super” com os quatro argumentos? E depois o “super do “possui_reimpressao”? Como ele vai saber, qual dos “super” é para chamar? Como definir a ordem de chamada?

[02:58] Eu quero falar do produto? Eu quero falar do impresso? Como eu posso fazer isso? Faço tudo de uma vez? O que eu faço?

[03:06] Toda vez que eu incluo um módulo dentro da minha classe em “Ruby”, na hora que eu tenho que chamar um método, seja o método normal ou super, existe uma ordem bem definida de como ele vai procurar esse método.

[03:19] Ele procura no módulo que eu acabei de incluir, e em outros módulos depois. Essa ordem é bem definida, mas ele só consegue acessar o último que eu incluí através dessa regra.

[03:35] Nesse caso, eu posso ir no módulo “impresso” e receber todos os parâmetros que eu quero receber, e chamar o “super”, com os quatro parâmetros: título, preço, ano de lançamento e editora.

[03:46] Mas quem é o “super”? Vamos dar uma olhada no caso de ter classe “livro”. O impresso “initialize” vai chamar o super, que no caso do livro, é o super que foi definido no produto. Quando rodamos, dá tudo certo. Só que o que acontece se eu remover o módulo produto?

[04:03] Quando tentamos executar o código, ele vai procurar o “super” que é sem argumento, e dá erro. Então quer dizer: começa a ficar tudo muito atrelado. No momento em que eu estou adicionando diversos módulos na minha classe, eu começo a ter um acoplamento entre os módulos que eu adiciono.

[04:18] Um tem que passar a conhecer como o outro funciona, pra não atrapalhar o funcionamento do outro. O acoplamento entre eles é muito alto. Na prática, tentamos usar módulo quando se comportamos como algo. E tentamos usar a herança de classe quando somos algo.

[04:40] No nosso caso, nós somos produtos. Então eu vou tirar esse construtor aqui do módulo, para evitar toda essa bagunça, vou falar que o livro é um produto, vou tirar o “possui_reimpressao” da maneira que estava antes, tradicional, atribuindo a variável membro.

[04:56] Mudo o nosso módulo “produto”, para “class Produto”. Vou fazer agora as outras classes também herdarem de produto, ao invés de incluírem um módulo. Em revista, vamos incluir também “Impresso”. Como produto, precisa de “require_relative “impresso””, e revista também vai precisar.

[05:24] Trabalhar com módulos e construtores, pode ser uma coisa um pouco perigosa, porque você está dizendo bastante sobre como esse módulo vai ser utilizado em relação a outros módulos dentro de uma classe.

[05:41] E é difícil conhecer todos os módulos e todas as maneiras possíveis que um módulo pode ser utilizado. Por isso, em relação a construção de um objeto, é comum trabalharmos na herança das classes, e não através da inclusão de módulos.

[05:56] Por fim, evitamos utilizar módulos que mexem com o estado interno de um objeto, porque, novamente a inclusão de diversos módulos pode gerar um efeito estranho.

[06:08] Uma boa prática na utilização de módulos, é evitar que eles acessem essas variáveis membro, permitindo que acesse somente métodos públicos dentro da própria classe onde ele vai ser incluído.