01

Modelos anêmicos e encapsulamento

Transcrição

[00:00] Observe que alguns livros antigos foram reimpressos, enquanto outros livros antigos não foram reimpressos. E é claro, dependendo da raridade do livro, nós queremos colocar um preço diferente.

[00:14] Então, com o passar do tempo a política de descontos da nossa empresa muda e temos que mudar a lógica que calcula o preço.

[00:25] Tudo bem. No nosso caso, o que queremos fazer é: se o ano de lançamento for menor do que 2006 e possui uma reimpressão, nós vamos dar 10% de desconto.

[00:33] Se é menor do que 2006 e não possui reimpressão, nós só damos 5% de desconto. Se o livro é mais recente, depois de 2006 e antes de 2010, damos 4% de desconto, e caso o contrário não damos desconto nenhum.

[00:49] Como podemos mudar essa regra de cálculo de preço? Vamos no nosso método calcula preço, apagamos o cálculo anterior, antigo, que não faz mais sentido. E vamos primeiro criar a variável que define se tem reimpressão ou não. Isso é, possui reimpressão.

[01:10] No caso do nosso livro de algoritmos, tem reimpressão sim. Vamos implementar agora o método: se o ano de lançamento for menor que 2006, se ele possuir reimpressão, eu vou retornar 10% de desconto.

[01:20] Se ele não possuir reimpressão, 5% de desconto. Se o ano de lançamento for maior que 2006, e menor ou igual a 2010, 4% de desconto. Caso contrário, nenhum desconto.

[01:35] Claro, estamos em “ruby”. E reparem que todos esses “return” podem ser removidos. Porque é a única coisa que cada um desses “else” e “if” fazem.

[01:42] Para o nosso livro de algoritmos, que é um livro antigo, e que possui reimpressão, temos que ter 10% de desconto. Por isso, quando executamos o programa, o preço do nosso livro é 90. Vamos imprimir também, se ele possui reimpressão ou não. Como fazemos para acessar essa variável?

[02:03] No caso de valores booleanos, é comum o ponto de interrogação no final. Isso é, “possui_reimpressao?”. Criamos o método, executamos a aplicação novamente, está tudo funcionando.

[02:15] Vamos agora trabalhar com o estoque de livros. Então além do livro de algoritmos que nós já temos, vamos adicionar um novo livro de arquitetura.

[02:24] Então criamos a variável “arquitetura=Livro.new(Introdução a Arquitetura e Design de Software”, atribuímos o seu preço, que é 70 reais, seu ano de lançamento, que foi 2011, e se possui reimpressão, (true).

[02:37] Ou seja, ele possui uma reimpressão. Vamos criar agora o nosso estoque. Então agora criamos nossa array de estoque e adicionamos nossos livros de algoritmo e arquitetura.

[02:47] E vamos adicionar mais dois livros em nosso estoque: “estoque << Livro.new(“The Pragmatic Programmer”, 100, 1999)” seu preço, que é 100 reais, seu ano de lançamento, que é 1999, e ele possui reimpressão, então true.

[03:04] Um novo livro, “Programming Ruby”, seu preço que é 100, seu ano de lançamento, que é 2004, e também possui reimpressão, true. Agora nós queremos imprimir os livros do nosso estoque em um formato que outro sistema possa

interpretar.

03:23] Então queremos imprimir em cada linha, o título do livro e o ano do lançamento: “Algoritmos, 1998”, “Introdução a Arquitetura e Design de Software, 2011”, “The Pragmatic Programming, 1999” e “Programming Ruby, 2004”.

[03:52] Então vamos percorrer a nossa array de livros, ou seja, fazer a interação e imprimir em cada linha o “#{livro.titulo} ,#{livro.ano_lancamento}”.

[04:02] Apagamos essa parte que não será utilizada, mas não podemos deixar esse código voando em nosso arquivo. Então vamos extraí-lo para um método, chamado “exporta_csv”. Chamamos o método “exporta_csv(estoque)”.

[04:27] Agora, vamos organizar um pouco o nosso código, passando esse método pra cima. Executamos o nosso programa e tudo está funcionando bem, recebemos a saída esperada.

[04:37] Dado o nosso estoque, queremos saber agora os livros mais baratos que um determinado valor. 100 reais, 200 reais, então podemos criar um método chamado “mais_baratos_que(valor)”.

[04:54] Queremos então que “livro.preco<= valor”, então precisamos passar em cada livro do nosso estoque, aplicando essa regra. Agora vamos adicionar o parâmetro estoque, e selecionar em nosso estoque os livros que correspondem a essa regra.

[05:24] Ou seja, “estoque.select do IlivroI livro.preco <= valor”. Vamos criar o atributo baratos, que vai receber os livros mais baratos que 80 reais em nosso estoque, por exemplo. “baratos= mais_baratos_que(estoque, 80)”.

[05:47] Então, para cada um desses livros que a gente receber, precisamos imprimir o título do livro: “baratos.each do IlivroI”. Imprimimos “livro.título”. Vamos organizar o nosso código, movendo o método para cima, junto com os demais.

[06:10] Executamos nossa aplicação novamente e tudo está funcionando. Recebemos o livro “Introdução a Arquitetura e Design de Software”, pois ele é abaixo de 80 reais.

[06:18] Repare que o estoque é uma parte muito importante da nossa aplicação, nós estamos criando vários métodos com funcionalidades relacionadas ao estoque. Então poderemos criar a “class Estoque”. Críamos o atributo livros, na classe estoque, e em seu construtor, inicializamos ele com uma array.

[06:40] Agora podemos cria um novo estoque: estoque= “Estoque.new”, e adicionar os livros através de “estoque.livros <<algoritmos<<arquitetura”, “estoque.livros<<Livro.new<< “The Pragmatic Programmer”” e assim por diante.

[07:02] Podemos agora mover o nosso método “exporta_csv”, para dentro da nossa classe estoque. Então alteramos a sua chamada para “estoque.exporta_csv”, e movemos o método para dentro da classe estoque.

[07:23] Podemos retirar o parâmetro estoque, e agora ao invés de iterar dentro de uma array de estoque, vamos iterar dentro de uma array de livros.

[07:29] Para verificar o funcionamento de nosso sistema, vamos comentar o método “mais_barato_que”. Executamos o código novamente, e tudo está funcionando. Descomentamos o código. Vamos mover agora o método “mais_baratos_que” para dentro do nosso estoque.

[07:50] Então alteramos a nossa chamada para “estoque.mais_baratos_que”, retiramos o parâmetro estoque, pois não será mais necessário, e agora também vamos iterar dentro do array de livros.

[08:02] “Livros.select do |livro|”, quando livro.preco for menor ou igual ao valor. Executamos nosso código novamente, e tudo continua funcionando.

[08:21] Recebemos o nosso livro, “Introdução a Arquitetura e Design de Software”, mas como descobrir a quantidade de livros dentro do meu estoque?

[08:22] Ao invés de acessar diretamente a variável livros, vamos criar um método que represente essa nossa necessidade. Criamos o nosso método total, que vai nos devolver o tamanho da nossa array de livros, ou seja: “livros.size”.

[08:42] Pra criar uma nova instância de livros, é preciso passarmos novos argumentos, seu título, preço, ano de lançamento, e se possui reimpressão. Mas o que acontece se tentarmos adicionar um livro nulo no nosso estoque?

[08:58] “estoque.livros<<nil”, executamos a nossa aplicação e recebemos uma exception. Pra evitar que isso aconteça, podemos criar o método adiciona, que será o responsável por adicionar novos livros em nosso estoque.

[09:13] Então alteramos a chamada do método para “estoque.adiciona livro” e nossa em nossa classe estoque, implementamos esse método “adiciona”: “def adiciona (livro)”, e adicionará esse livro se ele for um livro válido. Ou seja: “@livros<< livro if livro.”

[09:41] Testamos o nosso método, adicionando um livro nulo. Executamos nossa aplicação novamente, e tudo está funcionando. Adicionar um livro ao estoque agora é um feito invocando um método “adiciona”.

[09:51] Vamos alterar então a forma que estamos adicionando os nossos livros ao estoque, como nosso método adiciona só recebe um livro, alteramos a forma que estamos adicionando o livro de arquitetura: “estoque.adiciona arquitetura”. Executamos nosso programa novamente, e agora tudo está funcionando.

[10:17] Retiramos o atributo attr_reader de livros, pois assim estamos garantindo que no momento de adicionar um novo livro, será utilizado o nosso método de adiciona, e não o método “adiciona” da classe array, que é utilizado internamente pelo estoque.

[10:30] Vamos alterar o nosso método “exporta_csv”, pra que ele também apresente o preço do livro. Executamos a aplicação, e recebemos a resposta esperada: título, ano e preço do livro.

[10:44] Observe que boa parte da funcionalidade do nosso método “exporta_csv”, consiste em fazer chamadas para uma instãncia de livros que é passada como parâmetro para o bloco de each. Poderemos encapsular o acesso aos dados de um livro, para que o próprio livro possa possuir a habilidade de se apresentar no formato csv.

[11:07] Vamos criar em nossa classe livro, o método “to_csv”, que vai acessar diretamente seus atributos, e nos devolver o texto formatado com vírgulas. Executamos nossa aplicação novamente, e tudo está funcionando.

[11:26] Agora, a nossa classe estoque não tem mais acesso as propriedades de um livro. Quando ela precisa, diz para o livro fazer algo. Nesse caso, devolver as informações na forma csv: “to_csv”.

[11:36] Ou seja, nosso método agora utiliza o princípio “tell don’t ask”. Ele diz para o livro fazer alguma coisa, e não pede.

[11:44] O encapsulamento é importante, não só para que fique claro qual objeto tem responsabilidade sobre qual informação, mas também para tornar os objetos menos dependentes ou acoplados um dos outros.

[11:55] Pois deixamos de ficar acoplados ao funcionamento interno de um estoque ou de um livro, e passamos a nos comunicar. A ficar acoplados somente a interface pública, externa de um objeto.