Checkout
(Dificuldade: Média)

23/05/2017

No comércio, no geral, vemos diversas regras arbitrárias na definição de preços e várias formas de promoções. Um exemplo clássico que pode ser encontrado em feiras ou supermercados: uma maçã custa 50 centavos, mas três custam R$ 1,30. Outro bem comum: leve 3, pague 2.

Em uma feira, onde não há qualquer informatização, o próprio comerciante reconhece as promoções e aplica o preço correto. Mas em um sistema automatizado que recebe apenas preços unitários, isso não é possível. Afinal, teriam problemas no fechamento do caixa ou no registro do estoque.

A rede de Supermercados Rocambole está atualmente utilizando um desses sistemas simples, mas vem tendo uma queda de vendas. Para tentar aumentar as vendas, querem criar novas promoções com seus produtos, mas se sentem limitados por seu sistema de caixa. Portanto, contrataram você para desenvolver uma nova solução que os permita não só cadastrar preços unitários aos produtos, mas também criar preços especiais de acordo com a quantidade comprada. Para convencê-los que seu sistema dará suporte a essas promoções, você deve desenvolver um protótipo demonstrando isso.

1 Tarefa

1.1 Especificação

Em supermercados, os itens são identificados por SKUs. Mas para simplificar o protótipo, usaremos letras do alfabeto como identificadores dos produtos. A tabela abaixo representa a tabela de preços dos produtos usados no protótipo (os preços estão em centavos).

Item	Preço unitário	Preço especial
A	50	3 por 130
B	30	2 por 45
C	20	leve 3, pague 2
D	15	

O caixa deve aceitar itens em qualquer ordem e mostrar o preço total a qualquer momento (o qual pode sofrer alteração à medida que itens são adicionados e promoções são detectadas). Além do preço total, o caixa deve também mostrar o desconto total. Outra regra é que deve ser possível remover um produto do caixa, desqualificando uma promoção, se necessário.

1.2 Entrada

O método de adição de produto que recebe uma String com seu identificador poderá ser chamado consecutivas vezes, podendo ter também chamadas do método de remoção de produto.

1.3 Saída

Dois métodos de saída devem existir: obter preço total e obter desconto total.

2 Testes

O funcionamento da solução deve ser garantida através de testes unitários. A seguir encontra-se um código em Java com as comparações e seus resultados. Essas são as comparações mínimas que devem ser feitas para garantir o funcionamento. Utilize esse código como base da implementação dos testes unitários.

Dica: Implemente os testes antecipadamente.

```
// Teste 1 (caixa zerado)
checkout.add("A");
checkout.getTotalPrice() == 50;
checkout.getTotalDiscount() == 0;
checkout.add("A");
checkout.getTotalPrice() == 100;
checkout.getTotalDiscount() == 0;
checkout.add("A");
checkout.getTotalPrice() == 130;
checkout.getTotalDiscount() == 20;
checkout.add("A");
checkout.getTotalPrice() == 180;
checkout.getTotalDiscount() == 20;
checkout.add("A");
checkout.getTotalPrice() == 230;
checkout.getTotalDiscount() == 20;
checkout.add("A");
checkout.getTotalPrice() == 260;
checkout.getTotalDiscount() == 40;
checkout.remove("A");
checkout.getTotalPrice() == 230;
checkout.getTotalDiscount() == 20;

// Teste 2 (caixa zerado)
checkout.add("D");
checkout.getTotalPrice() == 15;
checkout.getTotalDiscount() == 0;
checkout.add("A");
checkout.getTotalPrice() == 65;
checkout.getTotalDiscount() == 0;
checkout.add("B");
checkout.getTotalPrice() == 95;
```

```
checkout.getTotalDiscount() == 0;
checkout.add("A");
checkout.getTotalPrice() == 145;
checkout.getTotalDiscount() == 0;
checkout.add("B");
checkout.getTotalPrice() == 160;
checkout.getTotalDiscount() == 15;
checkout.add("A");
checkout.getTotalPrice() == 190;
checkout.getTotalDiscount() == 35;
checkout.remove("A");
checkout.getTotalPrice() == 160;
checkout.getTotalDiscount() == 15;
checkout.remove("B");
checkout.getTotalPrice() == 145;
checkout.getTotalDiscount() == 0;

// Teste 3 (caixa zerado)
checkout.add("C");
checkout.getTotalPrice() == 20;
checkout.getTotalDiscount() == 0;
checkout.add("C");
checkout.getTotalPrice() == 40;
checkout.getTotalDiscount() == 0;
checkout.add("C");
checkout.getTotalPrice() == 40;
checkout.getTotalDiscount() == 20;
checkout.add("C");
checkout.getTotalPrice() == 60;
checkout.getTotalDiscount() == 20;
checkout.remove("C")
checkout.getTotalPrice() == 40;
checkout.getTotalDiscount() == 20;
checkout.remove("C")
checkout.getTotalPrice() == 40;
checkout.getTotalDiscount() == 0;


// Teste 4 (caixa zerado)
checkout.add("C");
checkout.getTotalPrice() == 20;
checkout.getTotalDiscount() == 0;
checkout.add("B");
checkout.getTotalPrice() == 50;
checkout.getTotalDiscount() == 0;
checkout.add("B");
checkout.getTotalPrice() == 65;
checkout.getTotalDiscount() == 15;
checkout.remove("B");
checkout.getTotalPrice() == 50;
checkout.getTotalDiscount() == 0;
```

3 Considerações finais

Para avaliar a qualidade de seu código na solução desse problema, considere as seguintes reflexões:

- Se definirem outro tipo de promoção, além dessas duas, quão difícil seria adicioná-la à sua solução?

- Se definirem que múltiplas promoções para um mesmo produto podem coexistir, quão difícil seria dar suporte a isso de forma a considerar a promoção de maior desconto?