Code Kata 1 - Supermarket Prices

Dennis Mbae

8/23/2020

Contents

In this Kata

Some things in supermarkets have simple prices: a can of beans costs $0.65. Other things have more complex prices.

For example: - three for a dollar (so what's the price if I buy 4, or 5?) - $1.99/pound (so what does 4 ounces cost?) - buy two, get one free (so does the third item have a price?)

This kata involves no coding. The exercise is to experiment with various models for representing money and prices that are flexible enough to deal with these (and other) pricing schemes, and at the same time are generally usable (at the checkout, for stock management, order entry, and so on). Spend time considering issues such as:

- does fractional money exist? Yes. We store it in decimal places
- when (if ever) does rounding take place? To 2dp
- how do you keep an audit trail of pricing decisions (and do you need to)?
- are costs and prices the same class of thing?
- if a shelf of 100 cans is priced using "buy two, get one free", how do you value the stock?

Step 1 - Get a price catalog

I got a black friday catalog price from walmart in order to be able to see the different pricing options

https://www.bradsdeals.com/black-friday/walmart

I also went to amazon and did a random search and looked at the price filters to get an idea of how things might be priced differently

Step 2 - Check the different pricing options

- Normal prices - e.g. 100$
- Discounted prices - e.g. Buy for 328$ save 50$ so the original price is 378$
- Price based on a product differentiation e.g. iPhone with contract, without contract, 32gb, 64gb space etc, if you swap with an older product
- Buy X get Y free
- Buy X get Y% or Z$ off
- ~~Based on brand e.g. Gucci, Armani ...~~
- ~~Based on department e.g. Clothing, Sports and Fitness~~

Step 3 - A way to represent the prices

Environment/context to enable proper implemetation of pricing model

- Our database/list of prices should include a well-defined inventory and allow users to add their own products or request a new category of products to be created

  e.g.

  Products

  –Category: Technology(Unique field)

  —-Sub-category: Phones(Unique field)

  ——Product: Smartphones(Unique field)

  ———Specific: iPhones(Unique field)

- There will be several instances of a product to account for product differentiation

  e.g. iPhones:{iPhone:{version:X,...},iPhone:{version:6,...},...}

- Each product will have various properties that will be used to assign price. Will be user-defined but we can provide base properties based on previous products of the same category.

  e.g. iPhone:{version,contract,color,...}

Price Tracker

- We shall have a separate price tracker for the specific products in our inventory.
- We will have many definitions of price after speaking to SMEs about different ways to represent a price

  So it will contain various fields depending on the product metadata. Based on fields that differentiate the products in that Specific, we will generate a list of prices

```
iPhones:
{
  Prices:
    {
      version:{X},
      space:16GB,
      price:{original:100.00,real:50.00},
      special:{buyOneGetOneFree},
      ...
    },
    {
      version:{11},
      space:16GB,
      price:{original:200.00,real:66.66},,
      special:{30discount},
      ...
    }
}
```

- We will have a field called special which will be used to display any caveats e.g. buy one get two free and this will be algorithmically defined. These codes will be defined by users and the algorithms will be auto-generated.
- We will have special codes where a formula is applied to the price and the relevant field in the price object is adjusted

e.g.

```
def buyXGetYFree(product,X,Y):
  newPrice = product.price.original/(X+Y)
  product.setRealPrice(newPrice)

def discount(product,pct):
  newPrice = product.price.original * (pct/100)
  product.setRealPrice(newPrice)
```

Additional Price Tracker Rules

- All prices will be decimals(2dp). Truncated.
- All prices must be defined. A specific product must have an associated price.