06

Colocando em prática

Praticar? É pra já! :)

Python possui recursos de linguagens interessantíssimos que muitas vezes dispensam a aplicação de um design pattern. Vimos no primeiro capítulo que o padrão `Strategy` é facilmente implementado utilizado o paradigma funcional. Outros padrões realmente se fazem necessários, como foi o caso do `Chain of Responsability` , `Template Method` , entre outros. Já o padrão Builder, às vezes não é necessário, justamente pelo requinte que a linguagem Python nos dá.

Vamos resolver o problema de construção da nota fiscal abaixo utilizando recurso das linguagem Python. Esse ponto é importante para que você possa comparar a solução `build-in` do Python com o equivalente `Design Pattern` que mostraremos na resposta do exercício, assim que você o concluir. Chega de conversa, vamos revisitar o problema do capítulo.

Temos a classe `Nota_fiscal` que possui uma lista de `Itens` e outros atributos:

```
# -*- coding: UTF-8 -*-
# nota_fiscal.py
from datetime import date

class Item(object):

    def __init__(self, descricao, valor):
        self.__descricao = descricao
        self.__valor = valor

    @property
    def descricao(self):
        return self.__descricao

    @property
    def valor(self):
        return self.__valor

class Nota_fiscal(object):
    # os parâmetros opcionais devem ser os últimos
    def __init__(self, razao_social, cnpj, itens, data_de_emissao, detalhes):
        self.__razao_social = razao_social
        self.__cnpj = cnpj
        self.__data_de_emissao = data_de_emissao
        if(len(detalhes) > 20):
            raise NameError('Detalhes da nota não pode ter mais do que 20 caracteres')
        self.__detalhes = detalhes
        self.__itens = itens

    @property
    def razao_social(self):
        return self.__razao_social

    @property
    def cnpj(self):
        return self.__cnpj
```

```
    @property
    def data_de_emissao(self):
        return self.__data_de_emissao

    @property
    def detalhes(self):
        return self.__detalhes
```

Queremos poder construir uma nota fiscal, mas com as seguintes condições: `data_de_emissao` e `detalhes` são parâmetros opcionais no construtor, o restante é obrigatório. Lembre-se que Python possui parâmetros nomeados e que obrigatoriamente os parâmetros opcionais devem vir por último no construtor apenas.

Responda

INSERIR CÓDIGO		FORMATAÇÃO