04

Colocando em prática

Hora de praticar ainda mais um pouco! Vamos lá!

Não é raro queremos executar códigos com as mais diversas funcionalidades toda vez que algo importante acontecer com uma de nossas classes ou funções. Por exemplo, quando uma nota fiscal for criada, precisaremos enviar um e-mail para o financeiro, mandá-la para um sistema de impressão e ainda gravá-la no banco de dados. Em nosso código, como sabemos que uma nota fiscal é criada? Quando seu construtor é invocado! Sendo assim, nada mais justo do que executar todas essas ações dentro do construtor. Mas claro, colocando cada uma em sua respectiva classe para separarmos a responsabilidade do nosso código. Ok?

```
# -*- coding: UTF-8 -*-
# observadores.py

def envia_por_email(nota_fiscal):
    print 'enviando nota por e-mail...'

def salva_no_banco(nota_fiscal):
    print 'salvando no banco...'

def imprime(nota_fiscal):
    print 'imprimindo ...'
```

No módulo `observadores.py` , temos uma série de funções que devem ser chamadas quando uma nota fiscal for criada. Poderiam até ser classes, mas não houve necessidade.

Agora, basta importamos essa função e invocá-las no construtor da nossa nota fiscal:

```
# -*- coding: UTF-8 -*-
# nota_fiscal.py

from datetime import date
from observadores import (envia_por_email,
    salva_no_banco, imprime)

class Item(object):

    # código omitido

class Nota_fiscal(object):

    def __init__(self, razao_social, cnpj, itens, data_de_emissao=date.today(), detalhes=''):
        self.__razao_social = razao_social
        self.__cnpj = cnpj
        self.__data_de_emissao = data_de_emissao
        if(len(detalhes) > 20):
            raise NameError('Detalhe da nota superior à 20 caracteres!')
        self.__detalhes = detalhes
        self.__itens = itens

        # chamando as funções
```

```
            envia_por_email(self)
            salva_no_banco(self)
            imprime(self)

        # demais métodos omitidos

    if __name__ == '__main__':

        itens=[
            Item(
                descricao='ITEM A',
                valor=100
            ),
            Item(
                descricao='ITEM B',
                valor=200
            )
        ]

        nota_fiscal = Nota_fiscal(
            cnpj='012345678901234',
            razao_social='FHSA Limitada',
            itens=itens
        )
```

Executando nosso código no terminal:

```
python nota_fiscal.py
```

Excelente, nosso código funciona! Toda vez que uma nota fiscal for criada, executaremos cada um das nossas funções interessadas neste evento, isto é, nossos observadores. O problema dessa solução é que, toda vez que um novo observador for necessário, precisaremos alterar a classe `Nota_fiscal` ! Queremos algo mais flexível que não demande alterações toda vez que um novo observador for necessário.

Sua tarefa será chegar a este grau de flexibilidade aplicando o design pattern `Observer` a partir deste ponto. O vídeo e o texto explicado estão ao seu dispor, caso tenha dúvidas de como implementá-lo. Bom estudo!

Responda

INSERIR CÓDIGO		FORMATAÇÃO