07

Colocando em prática

Mais um design pattern para praticarmos, dessa vez o **chain of responsability**. Mas antes de começarmos a aplicá-lo, vamos ver o seguinte contexto.

Nossa classe `Orcamento` passa a ter uma lista de itens, sendo assim precisarmos alterá-la. Ela deve ficar assim:

```
# -*- coding: UTF-8 -*-
# orcamento.py
class Orcamento(object):

    def __init__(self):
        self.__itens = []

    # quando a propriedade for acessada, ela soma cada item retornando o valor do orçamento
    @property
    def valor(self):
        total = 0.0
        for item in self.__itens:
            total+= item.valor
        return total

    # retornamos uma tupla, já que não faz sentido alterar os itens de um orçamento
    def obter_itens(self):

        return tuple(self.__itens)

    # perguntamos para o orçamento o total de itens
    @property
    def total_itens(self):
        return len(self.__itens)

    def adiciona_item(self, item):
        self.__itens.append(item)

# um item criado não pode ser alterado, suas propriedades são somente leitura
class Item(object):

    def __init__(self, nome, valor):
        self.__nome = nome
        self.__valor = valor

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

    @property
    def nome(self):
        return self.__nome
```

Nosso problema é o seguinte: dado um orçamento, queremos aplicar um desconto apenas dentro de uma cadeia de descontos. Vamos abraçar o paradigma orientado a objetos criando o arquivo `descontos.py` com as classes que corresponde

aos descontos por cinco itens e mais de quinhentos reais:

```
# -*- coding: UTF-8 -*-
# descontos.py
class Desconto_por_cinco_itens(object):

  def calcular(self, orcamento):

    if(orcamento.total_itens > 5):
        return orcamento.valor * 0.1
    else:
      # se não segue a regra, o desconto é zero!
      return 0

class Desconto_por_mais_de_quinhentos_reais(object):

  def calcular(self, orcamento):

    if(orcamento.valor > 500):
      return orcamento.valor * 0.07
    else:
      # se não segue a regra, o desconto é zero
      return 0
```

Veja, apesar de estarmos utilizando o paradigma orientado a objetos, podemos resolver o problema da aplicação de desconto proceduralmente como no exemplo abaixo que cria a classe `Calculador_de_descontos` no arquivo `calculador_de_descontos.py`:

```
# -*- coding: UTF-8 -*-
# calculador_de_descontos.py
from descontos import Desconto_por_cinco_itens, Desconto_por_mais_de_quinhentos_reais

class Calculador_de_descontos(object):

    def calcula(self, orcamento):

        desconto = Desconto_por_cinco_itens().calcular(orcamento)
        if(desconto == 0):
            desconto = Desconto_por_mais_de_quinhentos_reais().calcular(orcamento)

        # mais um if que testa se desconto é 0 e aplica aplica o desconto

        return desconto

        # outras possíveis regras aqui

if __name__ == '__main__':

    from orcamento import Orcamento, Item

    orcamento = Orcamento()
    orcamento.adiciona_item(Item('Item A', 100.0))
    orcamento.adiciona_item(Item('Item B', 50.0))
    orcamento.adiciona_item(Item('Item C', 400.0))
```

```
calculador_de_descontos = Calculador_de_descontos()
desconto = calculador_de_descontos.calcula(orcamento)
print 'Desconto calculado : %s' % (desconto)
# imprime 38.5
```

Mais uma vez um código funcional, mas difícil de manter. Que em `Calculador_de_impostos`, seu método `calcula` possui uma série de `IF's`. E se um novo desconto aparecer? Precisaremos alterar essa classe!

É a partir desse ponto que seu trabalho começa. Com base no que aprendeu neste capítulo, melhore o design do código apresentado resolvendo o problema da cadeia de descontos aplicando o design pattern **chain of responsability**.

Responda

INSERIR CÓDIGO		FORMATAÇÃO