FOCO NO MERCADO DE TRABALHO

CLASSES E MÉTODOS EM PYTHON

Vanessa Cadan Scheffer

DIAGRAMA DE CLASSES E HERANÇA

Um diagrama de classes permite ilustrar o reúso de código através do mecanismo de herança onde uma classe herda os atributos e métodos de outra classe.

Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

DESAFIO

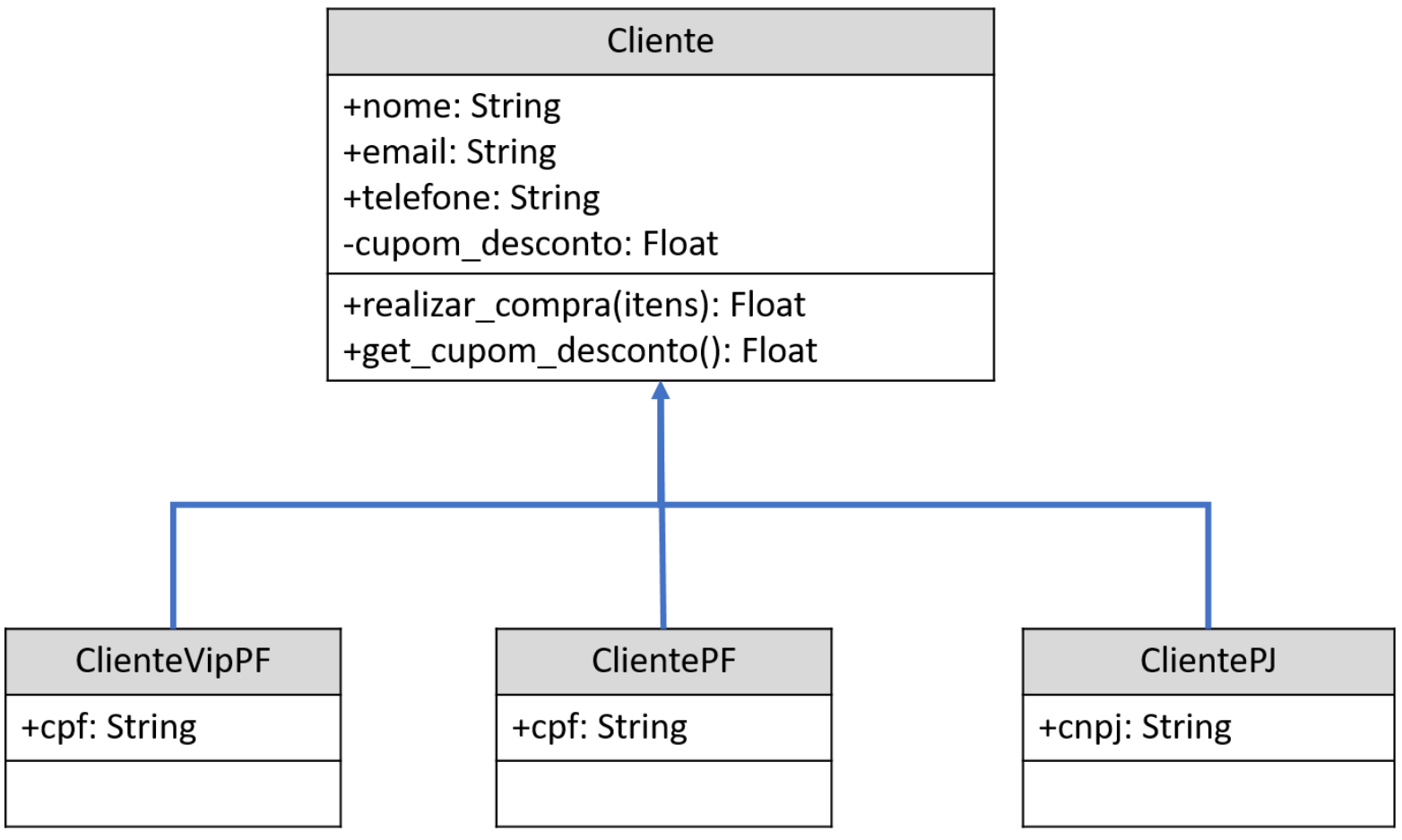
Como desenvolvedor em uma empresa de consultoria de software, você foi alocado para iniciar a implementação de um sistema de vendas. Nesta primeira etapa, os engenheiros de software fizeram um levantamento de requisitos e

identificaram que o sistema vai atender a três tipos de distintos de clientes: pessoas físicas (PF) que compram com assiduidade, pessoas físicas que compram esporadicamente e pessoas jurídicas (PJ). Os clientes VIPs podem comprar um número ilimitado de itens. Por sua vez, os clientes PF que não são VIPs, só podem comprar até 20 itens, e os clientes PJ podem comprar até 50 itens.

No levantamento de requisitos, também foi determinado que cada tipo de cliente terá um cupom de desconto que será usado para conceder benefícios nas compras. Os clientes VIPs terão um desconto de 20% a cada compra. Os clientes esporádicos (PF) terão um desconto de 5%. Por sua vez, os clientes com CNPJ terão um desconto de 10%. O cupom de desconto deverá ser encapsulado em um método.

Você recebeu a missão de implementar esse primeiro esboço do sistema, tarefa para cuja execução os engenheiros de software lhe entregaram o diagrama de classes representado na Figura 3.5. Agora é com você! Implemente a solução e apresente-a a equipe!

Figura 3.5 | Esboço inicial do diagrama de classes



Fonte: elaborada pela autora.

RESOLUÇÃO

Para este desafio, será necessário implementar quatro classes: uma base e três subclasses, que herdam os recursos. Todos os atributos que são comuns às classes devem ficar na classe-base. Por sua vez, os atributos específicos devem ser implementados nas classes-filhas.

A quantidade de itens que cada cliente pode comprar é diferente, razão pela qua método *realizar_compra* precisa ser sobrescrito em cada subclasse para atender essa especificidade. O cupom de desconto também varia de acordo com o tipo de cliente. Observe, a seguir, uma possível implementação para a solução.

In [22]:

```
class Cliente:
    def __init__(self):
        self.nome = None
        self.emil = None
        self.telefone = None
        self._cupom_desconto = 0

    def get_cupom_desconto(self):
        return self._cupom_desconto

    def realizar_compras(self, lista_itens):
        pass
```

In [23]:

```
class ClienteVipPF(Cliente):
    def __init__(self):
        super().__init__()
        self._cupom_desconto = 0.2

    def realizar_compras(self, lista_itens):
        return f"Quantidade total de itens comprados = {len(lista_itens)}"
```

In [24]:

```
class ClientePF(Cliente):
    def __init__(self):
        super().__init__()
        self._cupom_desconto = 0.05

    def realizar_compras(self, lista_itens):
        if len(lista_itens) <= 20:
            return f"Quantidade total de itens comprados = {len(lista_itens)}"
        else:
            return "Quantidade de itens superior ao limite permitido."
```

Ver anotações

In [25]:

```
class ClientePJ(Cliente):
    def __init__(self):
        super().__init__()
        self._cupom_desconto = 0.1

    def realizar_compras(self, lista_itens):
        if len(lista_itens) <= 50:
            return f"Quantidade total de itens comprados = {len(lista_itens)}"
        else:
            return "Quantidade de itens superior ao limite permitido."
```

In [26]:

```
cli1 = ClienteVipPF()
cli1.nome = "Maria"
print(cli1.get_cupom_desconto())
cli1.realizar_compras(['item1', 'item2', 'item3'])
```

```
0.2
```

Out[26]:

```
'Quantidade total de itens comprados = 3'
```

Implementamos as quatro classes necessárias. Veja que em cada classe-filha, determinados o valor do cupom de desconto logo após invocar o construtor da classe-base. Em cada subclasse também sobrescrevemos o método *realizar_compras* para atender aos requisitos específicos.

Conforme você pode notar, a orientação a objetos permite entregar uma solução mais organizada, com reaproveitamento de código, o que certamente pode facilit a manutenção e a inclusão de novas funcionalidades. Além disso, utilizar o diagrama de classes para se comunicar com outros membros da equipe tem grande valor no projeto de desenvolvimento de um software.

Utilize o emulador a seguir para testar a implementação e fazer novos testes!

Python 3.6

```
1  class Cliente:
2      def __init__(self):
3          self.nome = None
4          self.emil = None
5          self.telefone = None
6          self._cupom_desconto = 0
7
8      def get_cupom_desconto(self):
9          return self._cupom_desconto
10
11     def realizar_compras(self, list
12         pass
13
14
15 class ClienteVipPF(Cliente):
16     def __init__(self):
17         super().__init__()
```

Edit this code

Print output (drag lower right corner to resize)

Frames Objects

DESAFIO DA INTERNET

Ganhar habilidade em programação exige estudo e treino (muito treino).

Acesse a biblioteca virtual no endereço: http://biblioteca-virtual.com/ e busque pelo seguinte livro:

LJUBOMIR, P. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

Na página 301 do Capítulo 8 (*Programação orientada a objetos*) da referida obra,

você é convidado, no exercício 8.20, a criar uma classe chamada *minhaLista*. Essa classe deve se comportar como a classe list, exceto pelo método *sort*. Que tal tentar resolver esse desafio?!