Sistema de Gerenciamento de Lojas- Projeto de P.O.O.

Maria Gabriela de Paiva Sousa
Tainá Alcantara Alves Diniz

Objetivo do Programa:

Simular a gestão de vendas e estoque de produtos de uma empresa.

01 Classes

02 Objetos

03 Variáveis e Métodos de Instância

04 Herança e Sobrecarga de Operadores

05 Polimorfismo

06 Diagrama de Classes

Os atributos da classe Produto são:
codigo: número inteiro que representa o código do produto
nome: string que representa o nome do produto
preco: número de ponto flutuante que representa o preço do produto

Os métodos da classe Produto são:
init: método construtor que inicializa os atributos da classe
str: método que retorna uma string formatada com as informações do produto
get_codigo: método que retorna o código do produto.

```python
# Classe para produtos
class Produto:
    def __init__(self, codigo, nome, preco):
        self.codigo = codigo
        self.nome = nome
        self.preco = preco

    def __str__(self):
        return f"{self.nome} - R$ {self.preco:.2f}"

    def get_codigo(self):
        return self.codigo
```

Classe Cliente

Objetivo: Apresentar a classe Cliente e seus atributos e métodos

Definição da classe Cliente

Classe responsável por representar os clientes que realizam as compras.

Atributos:

nome: nome do cliente.

email: endereço de e-mail do cliente.

Métodos:

init: método construtor que recebe nome e e-mail do cliente e os atribui aos respectivos atributos.

str: método que retorna uma string formatada com o nome e e-mail do cliente.

```
14 # Classe para clientes
15 class Cliente:
16     def __init__(self, nome, email):
17         self.nome = nome
18         self.email = email
19 
20     def __str__(self):
21         return f"{self.nome} ({self.email})"
22 
```

A classe Venda é responsável por representar uma venda realizada para um cliente e é composta pelos seguintes atributos:

cliente: objeto da classe Cliente que representa o cliente que realizou a compra.
lista_produtos: lista de objetos da classe Produto que representa os produtos vendidos na transação.

Metodos:

init(self, cliente, lista_produtos): método construtor da classe, que recebe um objeto da classe Cliente e uma lista de objetos da classe Produto como parâmetros e os atribui aos respectivos atributos.

str(self): método que retorna uma string representando a venda realizada, contendo o nome do cliente e a lista de produtos vendidos com seus respectivos preços.

calcular_total(self): método que calcula e retorna o valor total da venda realizada, somando os preços de todos os produtos da lista de produtos.

```python
# Classe para vendas
class Venda:
    def __init__(self, cliente, lista_produtos):
        self.cliente = cliente
        self.lista_produtos = lista_produtos

    def __str__(self):
        return f"Venda para {self.cliente}:\n" + "\n".join([f"- {produto}" for produto in self.lista_produtos])

    def calcular_total(self):
        return sum([produto.preco for produto in self.lista_produtos])

```

A classe Estoque é responsável por armazenar as informações dos produtos disponíveis para venda na loja

Métodos:

- init(self, codigo, nome, preco, quantidade): método construtor da classe, responsável por inicializar os atributos da classe com os valores passados como parâmetros.
- str(self): método que retorna uma string que representa o objeto da classe em questão, formatada de uma maneira legível.
- atualizar_quantidade(self, quantidade): método que atualiza a quantidade disponível de um produto no estoque, levando em consideração a quantidade passada como parâmetro (se ela for negativa, é realizada uma verificação para garantir que não haja uma retirada maior que a quantidade disponível).

```python
# Classe para estoque (herda de Produto)
class Estoque(Produto):
    def __init__(self, codigo, nome, preco, quantidade):
        super().__init__(codigo, nome, preco)
        self.quantidade = quantidade

    def __str__(self):
        return f"{super().__str__()} ({self.quantidade} disponíveis)"

    def atualizar_quantidade(self, quantidade):
        self.quantidade += quantidade
```

Exemplo de Uso:

Criação de objetos das classes Produto, Cliente, Estoque e Venda com valores fictícios para seus atributos

Utilização dos métodos das classes, como calcular_total() da classe Venda e atualizar_quantidade() da classe Estoque

Impressão dos resultados na tela, como a lista de produtos da classe Venda após a adição de novos produtos e a quantidade atualizada de um produto na classe Estoque

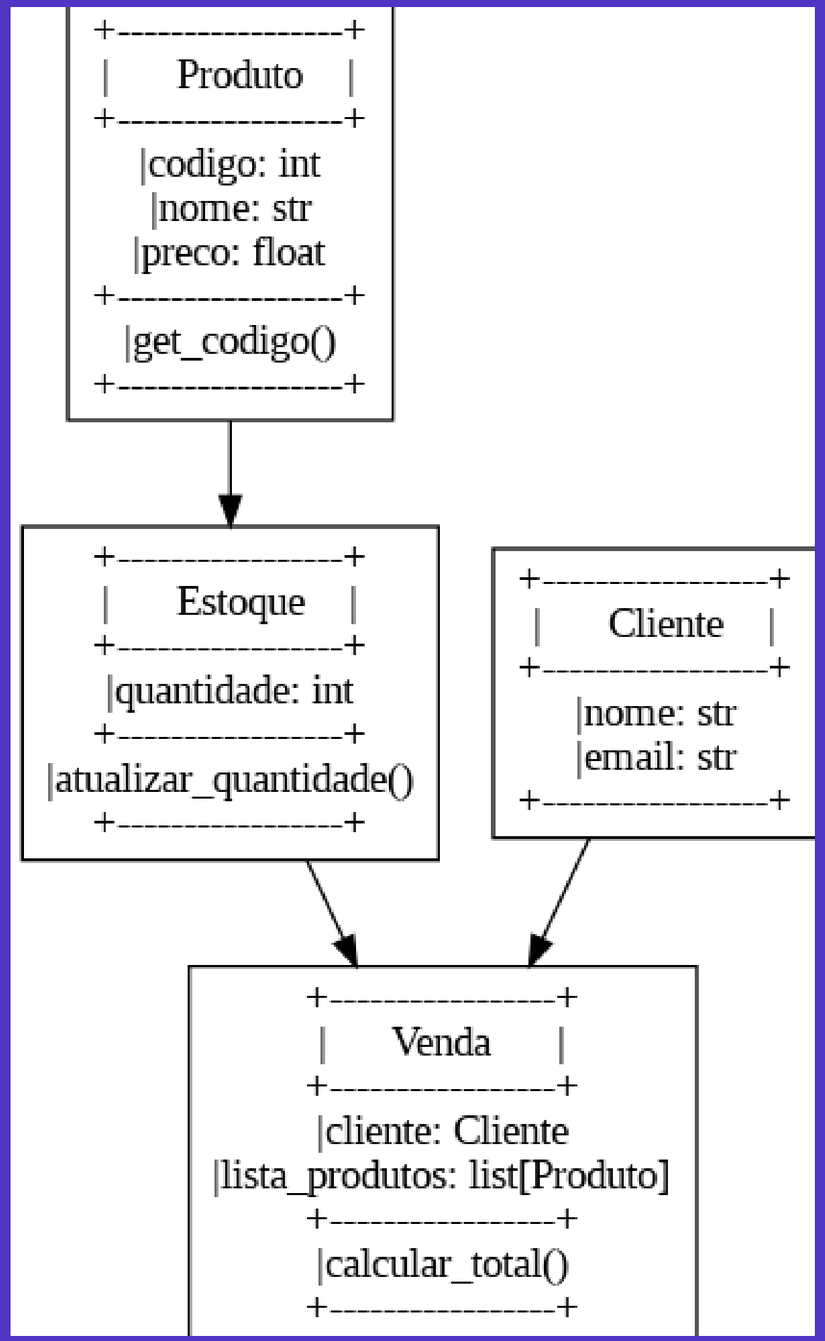
```python
# Exemplo de uso
p1 = Produto(1, "Celular", 1000.00)
p2 = Produto(2, "Notebook", 2000.00)
c1 = Cliente("João", "joao@gmail.com")
e1 = Estoque(1, "Celular", 1000.00, 10)
e2 = Estoque(2, "Notebook", 2000.00, 5)
v1 = Venda(c1, [p1, p2])

print(p1)
print(c1)
print(v1)
print(e1)

e1.atualizar_quantidade(5)
print(e1)
print(v1.calcular_total())
```

```
Celular - R$ 1000.00
João (joao@gmail.com)
Venda para João (joao@gmail.com):
- Celular - R$ 1000.00
- Notebook - R$ 2000.00
Celular - R$ 1000.00 (10 disponíveis)
Celular - R$ 1000.00 (15 disponíveis)
3000.0
```

Colab:


Produto
codigo: int
nome: str
preco: float
get_codigo()
Estoque
quantidade: int
atualizar_quantidade()
Cliente
nome: str
email: str
Venda
cliente: Cliente
lista_produtos: list[Produto]
calcular_total()