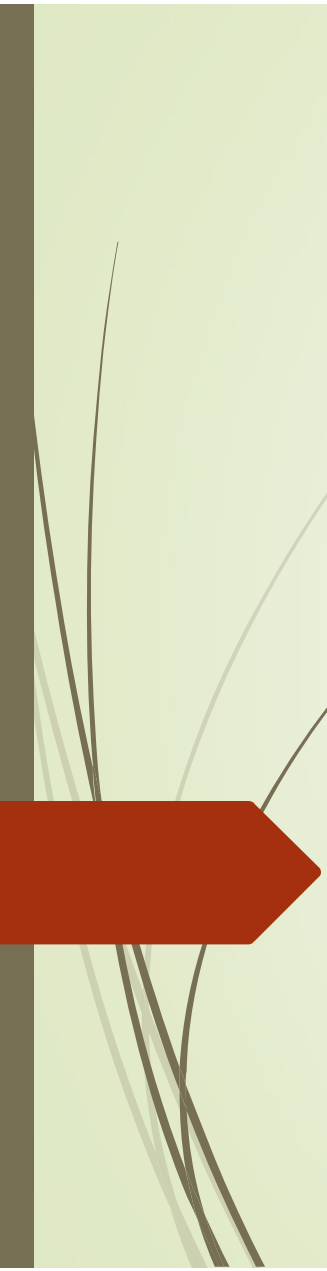


Implementando Classes em Python

Trabalhando com mais de uma classe

Agregação

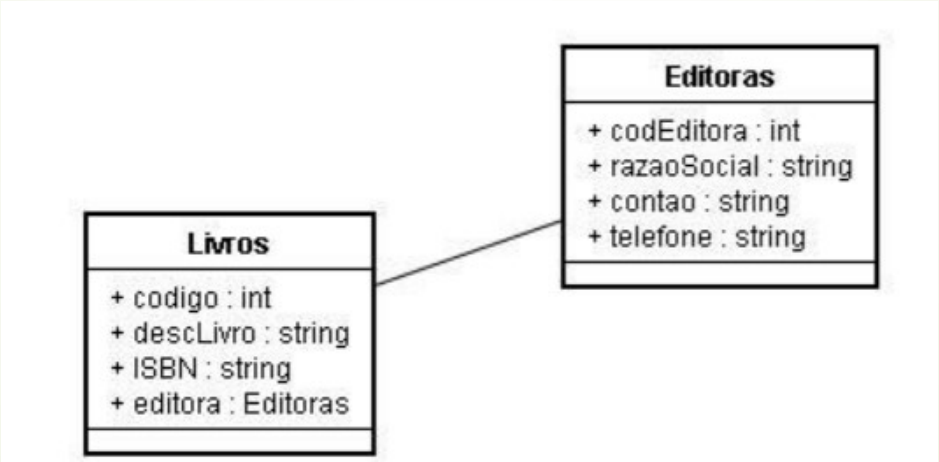
Composição

Associação

- Descreve um vínculo que ocorre entre classes (associação binária entre classes independentes);
  - Ou que uma associação seja compartilhada por mais de uma classe (associação ternária ou n-ária);
- Representamos as associações por meio de retas que ligam as classes envolvidas;
  - As retas podem ou não possuir setas nas extremidades indicando a navegabilidade da associação, ou seja, o sentido em que as informações são passadas entre as classes (não obrigatório);
  - Se não há setas, essas informações podem ser transmitidas entre todas as classes de uma associação.

Associação

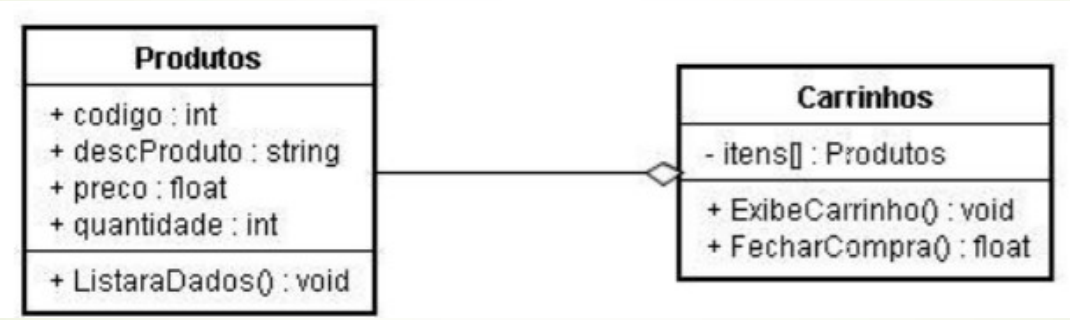
Exemplo



Agregação

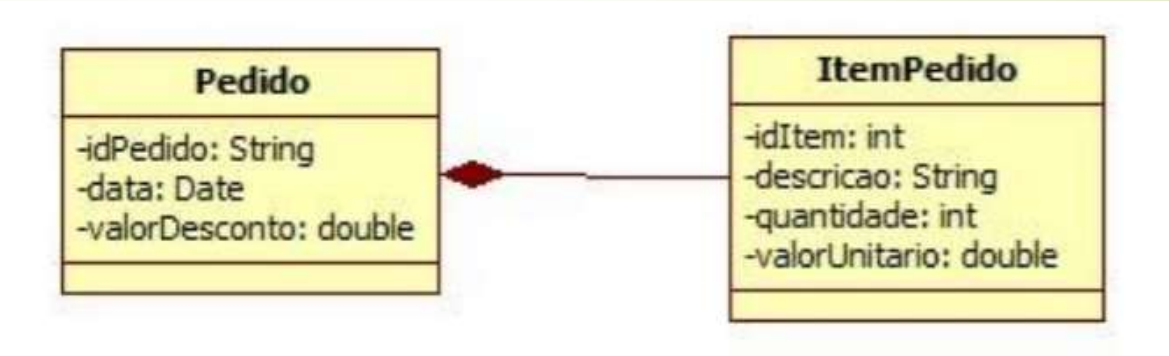
- Tipo especial de associação onde tenta-se demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte); conhecemos como todo/parte.
- Indica que uma das classes do relacionamento é uma parte ou está contida em outra classe.
- Semanticamente, representa: "consiste em", "contém", "é parte de".
  - Representamos uma agregação através de uma linha com um losango vazado na classe "TODO".

Agregação
Exemplo



Composição

- É uma variação da agregação.
- É uma agregação mais forte, onde a existência do Objeto-Parte não faz sentido se o Objeto-Todo não existir.
- Uma composição tenta representar também uma relação todo - parte. No entanto, o objeto-pai (todo) é responsável por criar e destruir suas partes.
- Em uma composição um mesmo objeto-parte não pode se associar a mais de um objeto-pai.



Exemplo:
Conta Bancária

Conta
número: int nome: String saldo: float limite: float
criar_conta(self,num,nome,saldo,limite) depositar(self,valor) retirar(self,valor) transferir(self,destino,valor) imprimir_extrato(self)

```python
class Conta:
    def __init__(self, numero, nome, saldo, limite):
        print("Inicializando uma conta...")
        self.numero = numero
        self.nome = nome
        self.saldo = saldo
        self.limite = limite
        print("Conta criada com sucesso!!")

    def deposita(self, valor):
        self.saldo += valor

    def retira(self,valor):
        if self.saldo < valor:
            print("Saldo insuficiente! Saque cancelado!")
        else:
            self.saldo -= valor

    def extrato(self):
        print(f"Conta: {self.numero}  -  Saldo: {self.saldo}"

    def transfere(self, destino, valor):
        if self.saldo > valor:
            self.saldo -= valor
            destino.saldo += valor
```

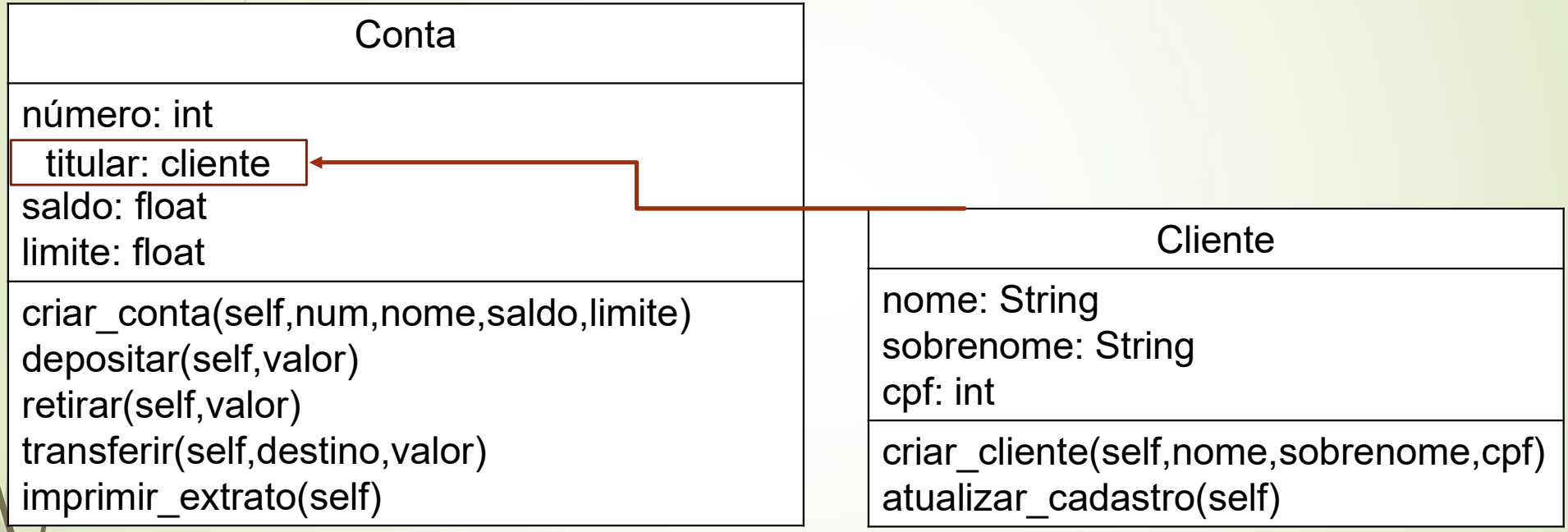
```python
from conta import Conta

conta1 = Conta(123,"Maria da Silva", 3000, 1000)
conta2 = Conta(124, "Joao da Siqueira",290,1000)

conta1.deposita(1000)
conta1.extrato()
conta1.retira(300)
conta1.extrato()
conta1.transfere(conta2,300)
conta1.extrato()
conta2.extrato()
```

```
>>> %Run conta_main.py
Inicializando uma conta...
Conta criada com sucesso!!
Inicializando uma conta...
Conta criada com sucesso!!
Conta: 123  -  Saldo: 4000
Conta: 123  -  Saldo: 3700
Conta: 123  -  Saldo: 3400
Conta: 124  -  Saldo: 590
```

Organizando o código...
- **agregar** o cliente à conta



```python
#cliente.py

class Cliente:
    def __init__(self,nome, sobrenome, cpf):
        self.nome = nome
        self.sobrenome = sobrenome
        self.cpf = cpf

class Conta:
    def __init__(self, numero, cliente, saldo, limite):
        self.numero = numero
        self.titular = cliente
        self.saldo = saldo
        self.limite = limite
```

Exemplo de Agregação

```
1  from cliente import Conta,Cliente
2
3  #quando uma conta é criada, precisamos passar um cliente como titular:
4  cliente = Cliente('João', 'Oliveira', '1111111111-1')
5  minha_conta = Conta('123-4', cliente, 120.0, 1000.0)
6
7  '''Aqui aconteceu uma atribuição, o valor da variável cliente é copiado
8  para o atributo titular do objeto ao qual minha_conta se refere.
9  Em outras palavras, minha_conta tem uma referência ao mesmo Cliente
10 que cliente se refere, e pode ser acessado através de
11 minha_conta.titular.
12 '''
```

```python
from cliente import Conta,Cliente

#quando uma conta é criada, precisamos passar um cliente como titular:
cliente = Cliente('João', 'Oliveira', '1111111111-1')
minha_conta = Conta('123-4', cliente, 120.0, 1000.0)

'''Aqui aconteceu uma atribuição, o valor da variável cliente é copiado
para o atributo titular do objeto ao qual minha_conta se refere.
Em outras palavras, minha_conta tem uma referência ao mesmo Cliente
que cliente se refere, e pode ser acessado através de
minha_conta.titular.
'''
print(minha_conta.titular.nome)
print(cliente.nome)
```

```
>>> %Run conta_main_2.py
João
João
```

- o exemplo anterior ilustra uma agregação porque o cliente existe independente da conta.

- Suponha que a conta contenha um histórico, contendo data de abertura e as transações. Neste caso o histórico somente irá existir se houver uma conta. O Histórico compõe a classe Conta.
  - Existe dependência, composição.

Histórico
data_abertura: data transações:[ ]
criar_historico() imprimir_histórico(self)

```python
import datetime

class Historico():
    def __init__(self):
        self.data_abertura=datetime.datetime.today
        self.transações=[]

    def imprime(self):
        print("data abertura: {}".format(self.data_abertura))
        print("transações: ")
        for t in self.transacoes:
            print("-", t)
```

modificações necessárias na classe Conta:

```
1  #conta_historico.py
2  import datetime
3  class Historico():
4      def __init__(self):
5          self.data_abertura = datetime.datetime.today()
6          self.transacoes=[]
7      def imprime(self):
8          print("data abertura: {}".format(self.data_abertura))
9          print("transações: ")
10         for t in self.transacoes:
11             print("-", t)
12
13 class Conta():
14     def __init__(self, numero, cliente, saldo, limite):
15         self.numero = numero
16         self.cliente = cliente
17         self.saldo = saldo
18         self.limite = limite
           self.historico = Historico()
```

```python
class Conta():
    def __init__(self, numero, cliente, saldo, limite):
        self.numero = numero
        self.cliente = cliente
        self.saldo = saldo
        self.limite = limite
        self.historico = Historico()

    def deposita(self, valor):
        self.saldo += valor
        self.historico.transacoes.append("depósito de {}".format(valor))

    def retira(self,valor):
        if self.saldo < valor:
            print("Saldo insuficiente! Saque cancelado!")
        else:
            self.saldo -= valor
            self.historico.transacoes.append("saque de {}".format(valor))

    def extrato(self):
        print(f"Conta: {self.numero}  -  Saldo: {self.saldo}")
        self.historico.transacoes.append(f"tirou extrato - saldo de {self.saldo}")

    def transfere(self, destino, valor):
        if self.saldo > valor:
            self.saldo -= valor
            destino.saldo += valor
            self.historico.transacoes.append(f"transferencia de {valor} para conta {destino.numero}")
```

Exercícios

- Implemente o código exemplo, crie uma lista para armazenar as contas e outra lista para armazenar os clientes do banco.

- Faça um menu onde o usuário possa definir a ordem das operações.
  - 1. Criar conta
  - 2. Operação de saque
  - 3. Operação de depósito
  - 4. Operação de transferência
  - 5. Extrato (mostrar o histórico das transações)
  - 6. Cadastrar cliente
  - 7. Sair

- Fonte:
- Apostila Caellum (Cap 7. Orientação a Objetos)
  - https://www.caelum.com.br/apostila-python-orientacao-objetos/orientacao-a-objetos/