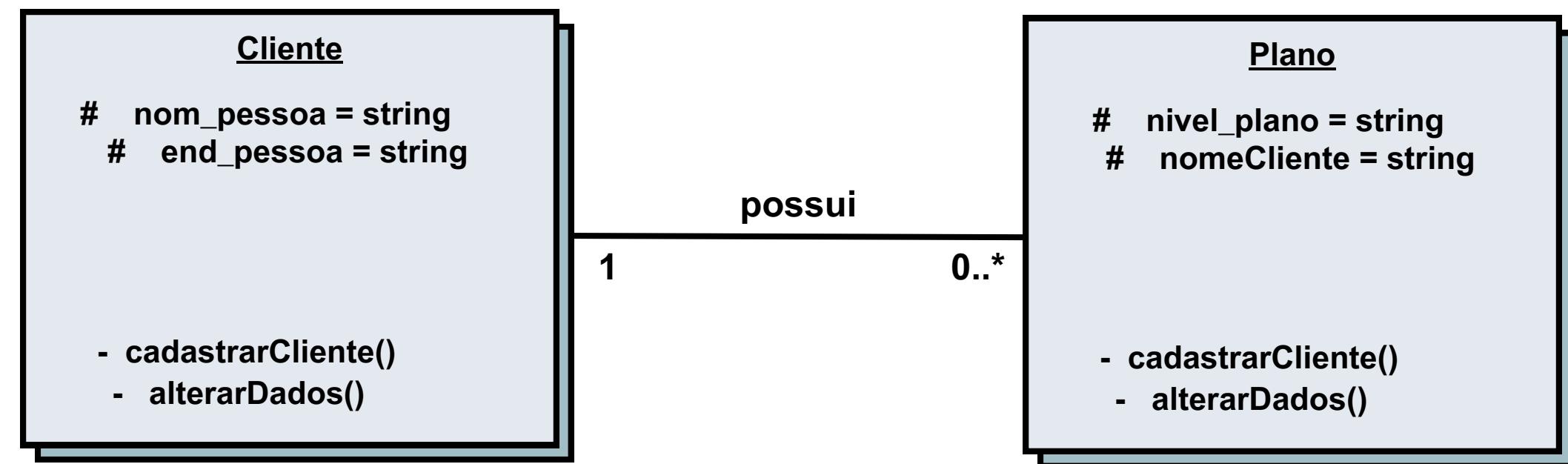
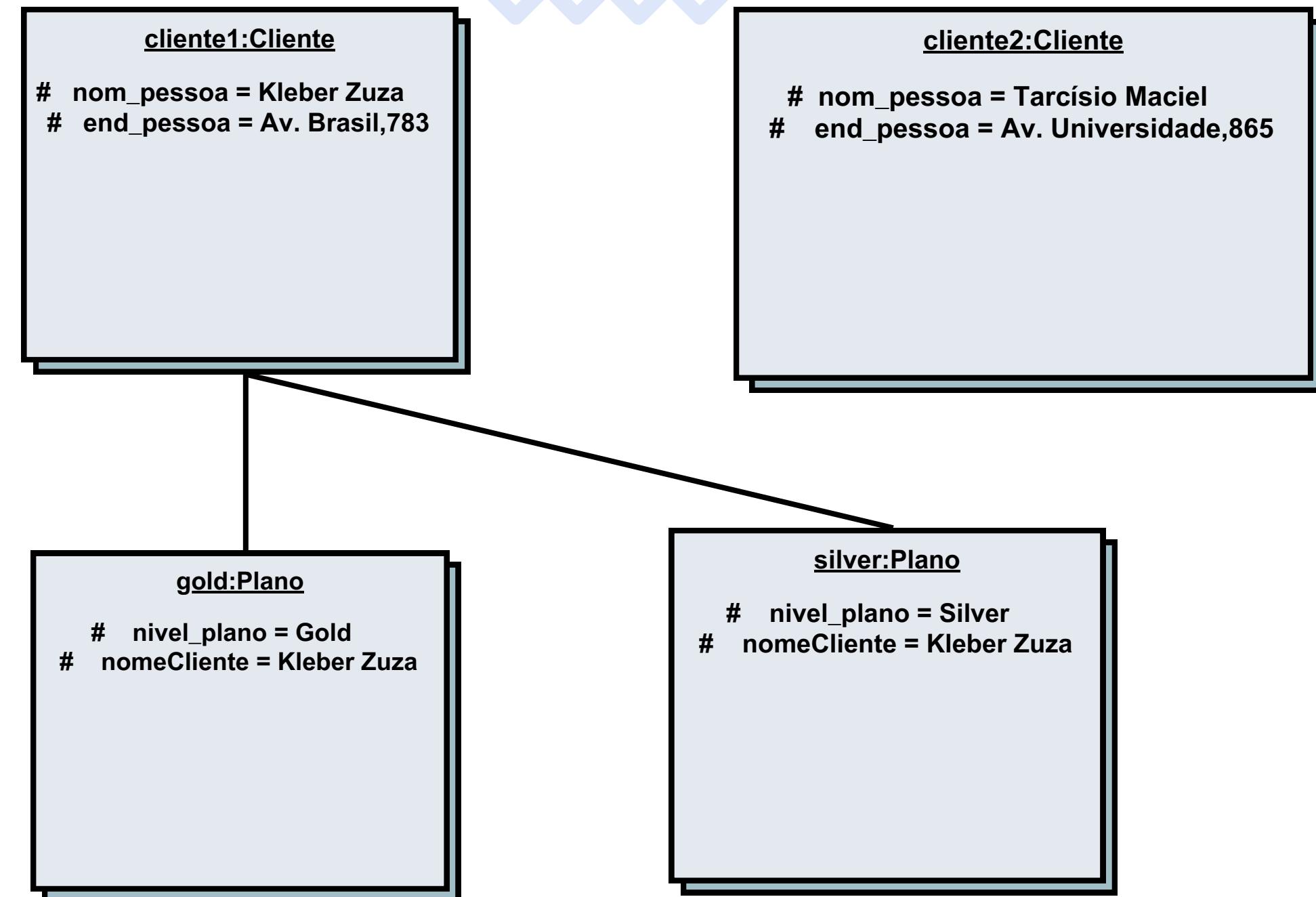


{Exemplo de Associação}



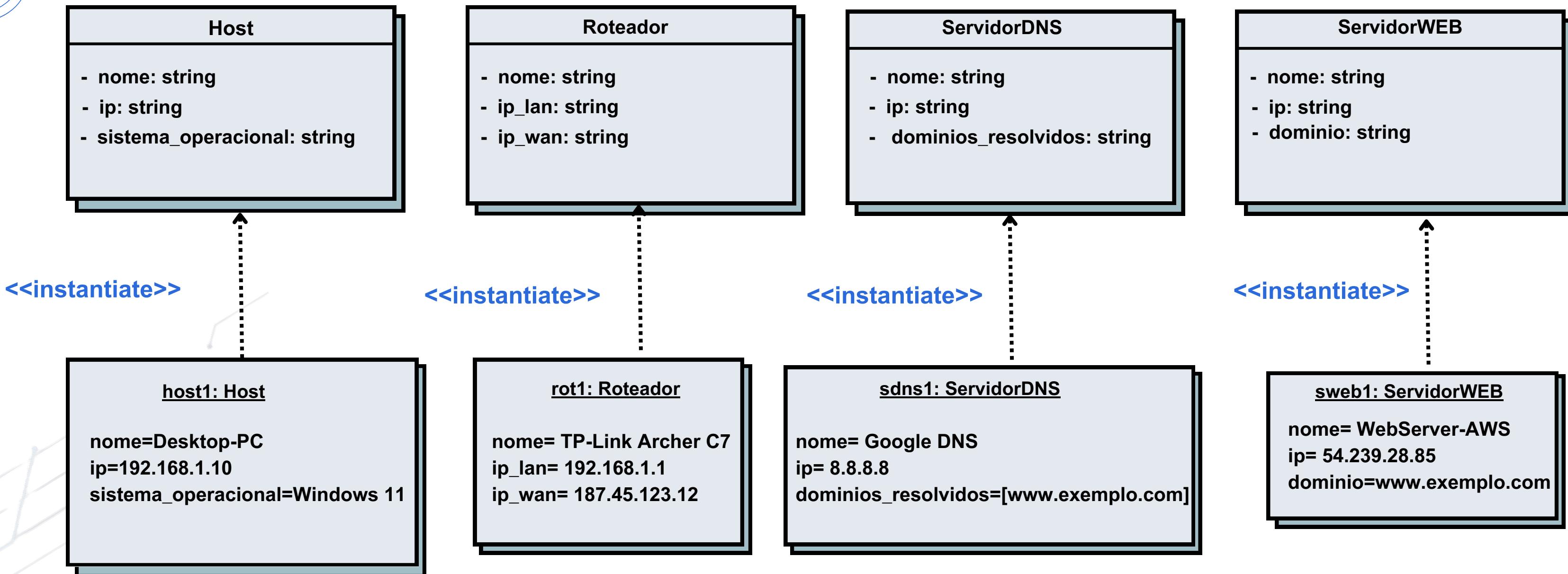
- Neste exemplo criamos uma secção de um diagrama de classes, o qual define duas classes(**Cliente** e **Plano**) que são relacionadas e tem multiplicidade.

{ Exemplo de Vínculo }

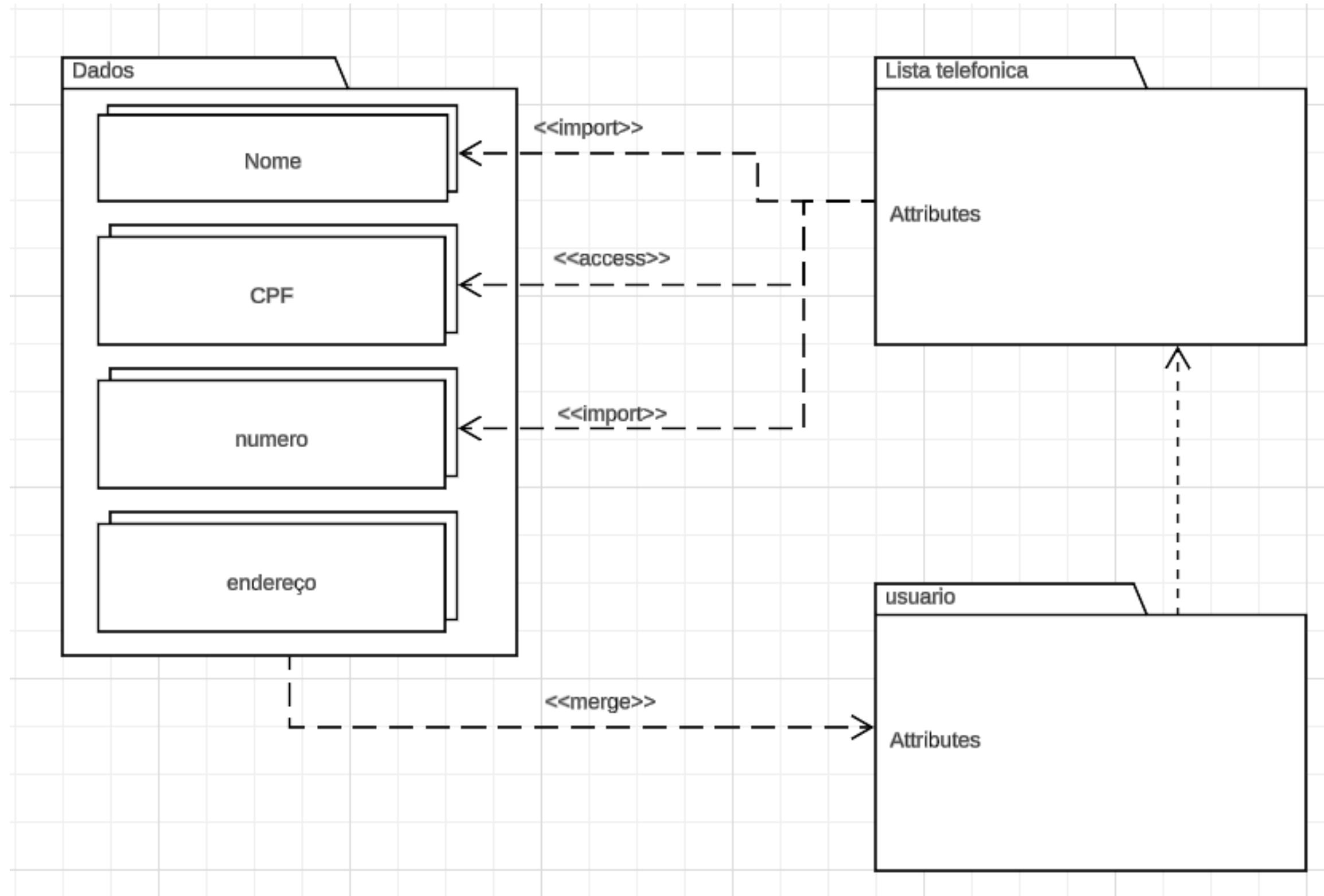


```
1  class Host:
2      def __init__(self, nome, ip, sistema_operacional):
3          self.nome = nome
4          self.ip = ip
5          self.sistema_operacional = sistema_operacional
6
7  class Roteador:
8      def __init__(self, nome, ip_lan, ip_wan):
9          self.nome = nome
10         self.ip_lan = ip_lan
11         self.ip_wan = ip_wan
12
13 class ServidorDNS:
14     def __init__(self, nome, ip, dominios_resolvidos):
15         self.nome = nome
16         self.ip = ip
17         self.dominios_resolvidos = dominios_resolvidos
18
19 class ServidorWeb:
20     def __init__(self, nome, ip, dominio):
21         self.nome = nome
22         self.ip = ip
23         self.dominio = dominio
24
25 # Instanciando objetos
26 host1 = Host(nome="Desktop-PC", ip="192.168.1.10", sistema_operacional="Windows 11")
27 roteador1 = Roteador(nome="TP-Link Archer C7", ip_lan="192.168.1.1", ip_wan="187.45.123.12")
28 servidorDNS1 = ServidorDNS(nome="Google DNS", ip="8.8.8.8", dominios_resolvidos=["www.exemplo.com"])
29 servidorWeb1 = ServidorWeb(nome="WebServer-AWS", ip="54.239.28.85", dominio="www.exemplo.com")
30
31 # Exibindo atributos
32 print(f"Host1: {host1.nome}, IP: {host1.ip}, SO: {host1.sistema_operacional}")
33 print(f"Roteador1: {roteador1.nome}, IP LAN: {roteador1.ip_lan}, IP WAN: {roteador1.ip_wan}")
34 print(f"ServidorDNS1: {servidorDNS1.nome}, IP: {servidorDNS1.ip}, Domínios: {servidorDNS1.dominios_resolvidos}")
35 print(f"ServidorWeb1: {servidorWeb1.nome}, IP: {servidorWeb1.ip}, Domínio: {servidorWeb1.dominio}")
```

{ Diagrama do Código }



{ Exemplo de Dependência }



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

    def __str__(self):
        return f"Cliente: {self.nome}, CPF: {self.cpf}"

class Plano:
    def __init__(self, nome: str, valor: float, franquia: int):
        self.nome = nome
        self.valor = valor
        self.franquia = franquia

    def __str__(self):
        return f"Plano: {self.nome}, Valor: R${self.valor:.2f}, Franquia: {self.franquia} GB"

from servico import Cliente, Plano
```



```
class Contrato:
    def __init__(self, cliente: Cliente, plano: Plano):
        self.cliente = cliente
        self.plano = plano

    def __str__(self):
        return f"{self.cliente} está vinculado ao {self.plano}."
```

```
from servico import Cliente, Plano
from contrato import Contrato

def executar_sistema():
    # Criando instâncias de Cliente e Plano
    cliente1 = Cliente("Ana Silva", "123.456.789-00")
    plano1 = Plano("Premium", 99.99, 50)

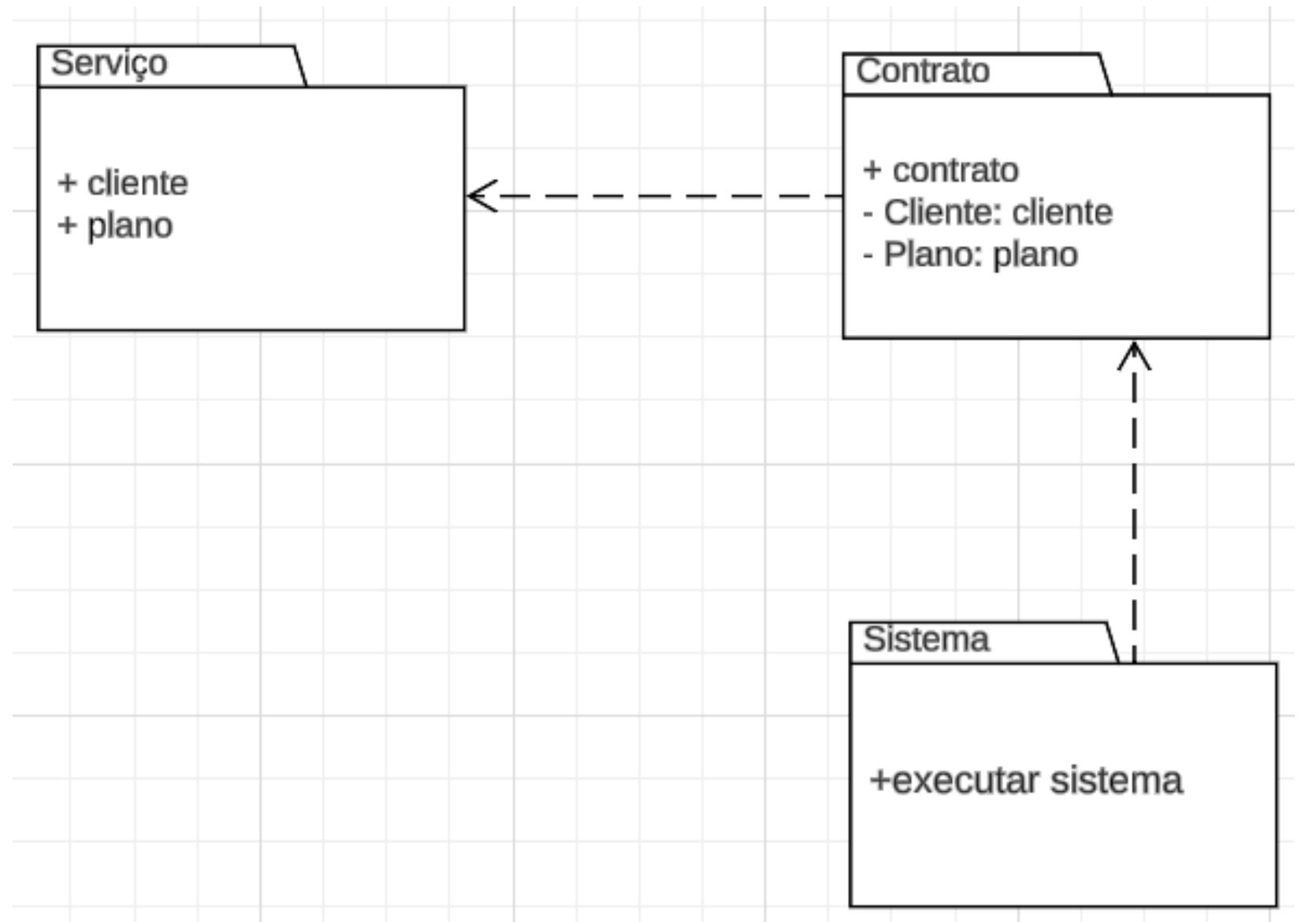
    # Criando Contrato
    contrato1 = Contrato(cliente1, plano1)

    # Exibindo informações do contrato
    print(contrato1)

if __name__ == "__main__":
    executar_sistema()
```

```
Cliente: Ana Silva, CPF: 123.456.789-00 está vinculado ao Plano: Premium, Valor: R$99.99, Franquia: 50 GB.
```

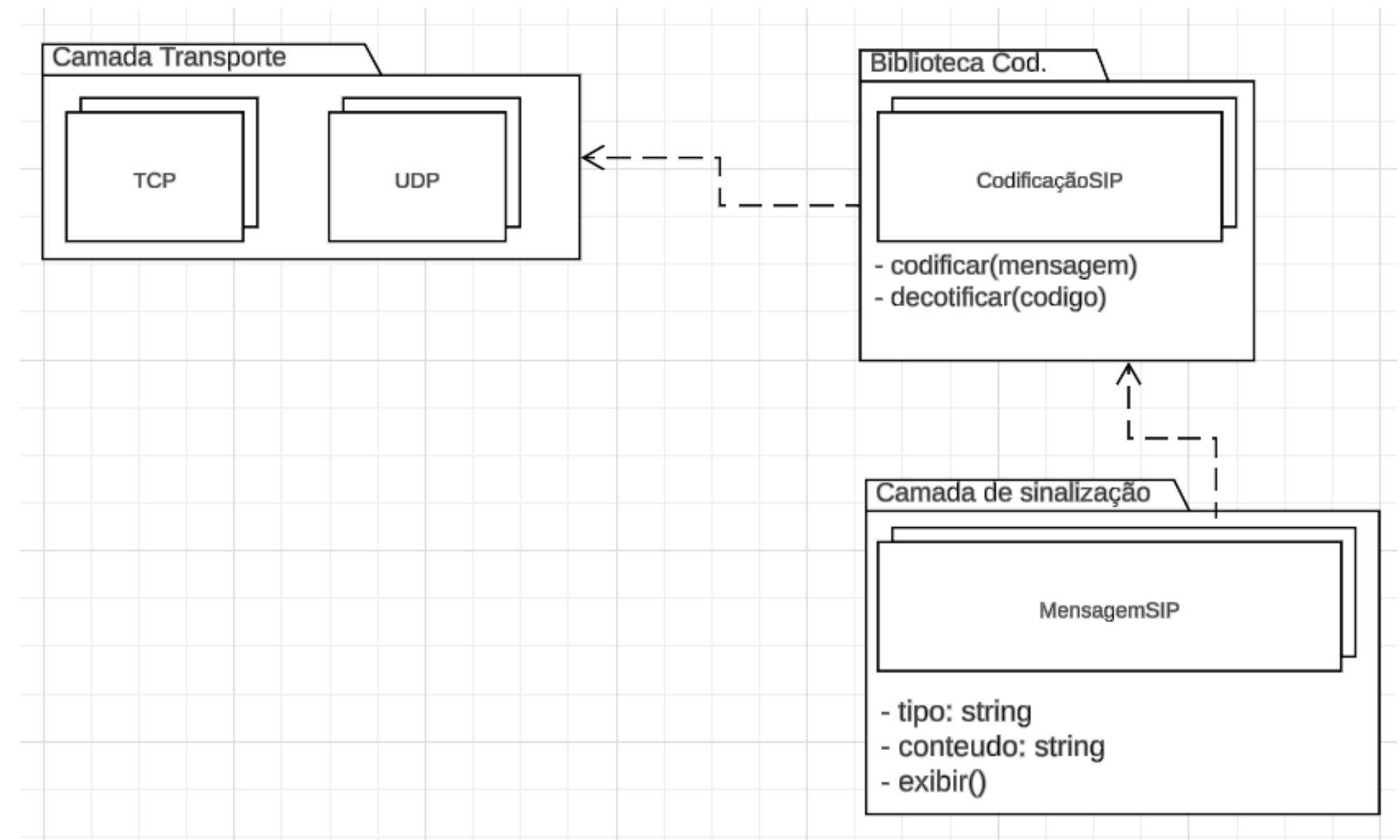
{ Diagrama do código }



```
class TCP:  
    def enviar_dados(self, dados):  
        print(f"TCP enviando: {dados}")  
  
    def receber_dados(self):  
        print("TCP recebendo dados")  
        return "dados recebidos via TCP"  
  
class UDP:  
    def enviar_pacote(self, pacote):  
        print(f"UDP enviando: {pacote}")  
  
    def receber_pacote(self):  
        print("UDP recebendo pacote")  
        return "pacote recebido via UDP"  
  
class MensagemSIP:  
    def __init__(self, tipo, conteudo):  
        self.tipo = tipo  
        self.conteudo = conteudo  
  
    def exibir(self):  
        return f"Mensagem SIP [{self.tipo}]: {self.conteudo}"
```

```
class CodificacaoSIP:  
    @staticmethod  
    def codificar(mensagem: MensagemSIP):  
        print(f"Codificando mensagem SIP: {mensagem.exibir()}")  
        return f"Código SIP de {mensagem.tipo}"  
  
    @staticmethod  
    def decodificar(codigo):  
        print(f"Decodificando código SIP: {codigo}")  
        return MensagemSIP("Resposta", "Decodificado com sucesso")  
  
tcp = TCP()  
udp = UDP()  
  
mensagem = MensagemSIP("INVITE", "Convidando usuário para sessão")  
codificada = CodificacaoSIP.codificar(mensagem)  
  
tcp.enviar_dados(codificada)  
recebido = tcp.receber_dados()  
  
resposta = CodificacaoSIP.decodificar(recebido)  
print(resposta.exibir())
```

```
Codificando mensagem SIP: Mensagem SIP [INVITE]: Convidando usuário para sessão  
TCP enviando: Código SIP de INVITE  
TCP recebendo dados  
Decodificando código SIP: dados recebidos via TCP  
Mensagem SIP [Resposta]: Decodificado com sucesso
```



BIBLIOGRAFIA

UML 2 - Uma abordagem prática - Gilleannes T.A
Guedes

Chat Blackbox: AI Code Generation, Code Chat, Code Search

Lucidchart | Diagramação com inteligência

