

Programação  
OO

||

---

||

# Parte II

---



UNIVERSIDADE  
**VILA VELHA**  
ESPÍRITO SANTO

# Dependências de classes

---

Entendido tudo até agora podemos conversar sobre

- relacionamento entre as objetos e a sua
- representação de navegabilidade...



# Implementação de relações

---

Em OO os objetos se relacionam entre si por meio da implementação de relações, tais como:

- Associação
- Agregação
- Composição

A decisão de qual relacionamento será usado é tomada durante a análise do sistema

Com base nessas relações podemos construir objetos compostos e mais elaborados...

- Vamos falar um pouco sobre esses relacionamentos..



UNIVERSIDADE  
**VILA VELHA**  
ESPÍRITO SANTO

# Implementação de relações – Associação

---

## Associações

- Uma associação representa que duas classes possuem uma ligação (link) entre elas, significando por exemplo que:
  - “A conhece um outro B”,
  - “A tem um B”,
  - “B é usado por A”
- Exemplos:
  - Uma Pessoa trabalha para uma Companhia,
  - Uma Companhia tem vários Escritórios,
  - Um sócio tem vários dependentes , etc....
- Podemos representar e implementar essa relação da seguinte forma...



# Implementação de relações – Associação

Relações de 1 para 1 bidirecional



**B "é usado por" A**  
e  
**A "é usado por" B**

**B "tem um" A**  
e  
**A "tem um" B**

```
class A{  
    ➡ private B b;  
    ...  
    public void setB(B aB){  
        b=aB;  
    }  
  
    public B getB(){  
        return b;  
    }  
    ...  
}
```

```
class B{  
    ➡ private A a;  
    ...  
    public void setA(A aA){  
        a=aA;  
    }  
  
    public A getA(){  
        return a;  
    }  
    ...  
}
```

Representado pela **linha sólida**



# Implementação de relações – Associação

## Relações de 1 para 2 bidirecional



**B "é usado por" A**  
e  
**A "é usado por" B**

**B "tem um" A**  
e  
**A "tem dois" B**

```
class A{  
    ➡ private B b1;  
    ➡ private B b2;  
    ...  
    public void setB1(B aB){  
        b1=aB;  
    }  
    public B getB1(){  
        return b1;  
    }  
  
    public void setB2(B aB){  
        b2=aB;  
    }  
}
```

```
class B{  
    ➡ private A a;  
    ...  
    public void setA(A aA){  
        a=aA;  
    }  
  
    public A getA(){  
        return a;  
    }  
    ...  
}
```



# Implementação de relações – Associação

## Relações de 1 para 1 unidirecional

**B “é usado por” A**



**A “tem um” B**

```
class A{  
    ➡ private B b;  
    ...  
    public void setB(B aB){  
        b=aB;  
    }  
  
    public B getB(){  
        return b;  
    }  
    ...  
}
```

```
class B{  
    ...  
}
```

Um relacionamento sem navegabilidade **implica** que ele pode ser lido de duas formas, isto é, em suas duas direções.

Utilizando a propriedade de navegabilidade, podemos restringir a forma de ler um relacionamento.

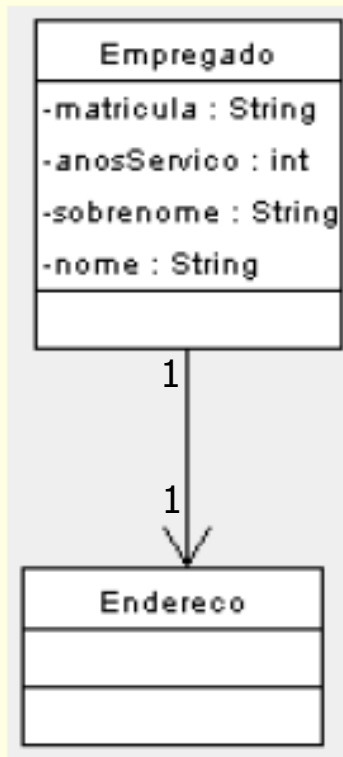
A seta no final da associação indica que a navegabilidade será de A para B





# Implementação de relações – Associação

## Relações de 1 para 1 unidirecional



```
public class Empregado {
    private String matricula;
    private int anosServico;
    private String sobrenome;
    private String nome;
    ➡ private Endereco endereco;
    . . .
}

public class Endereco {
    . . .
}
```

**Empregado “tem um” Endereço**



# Implementação de relações – Agregação

---

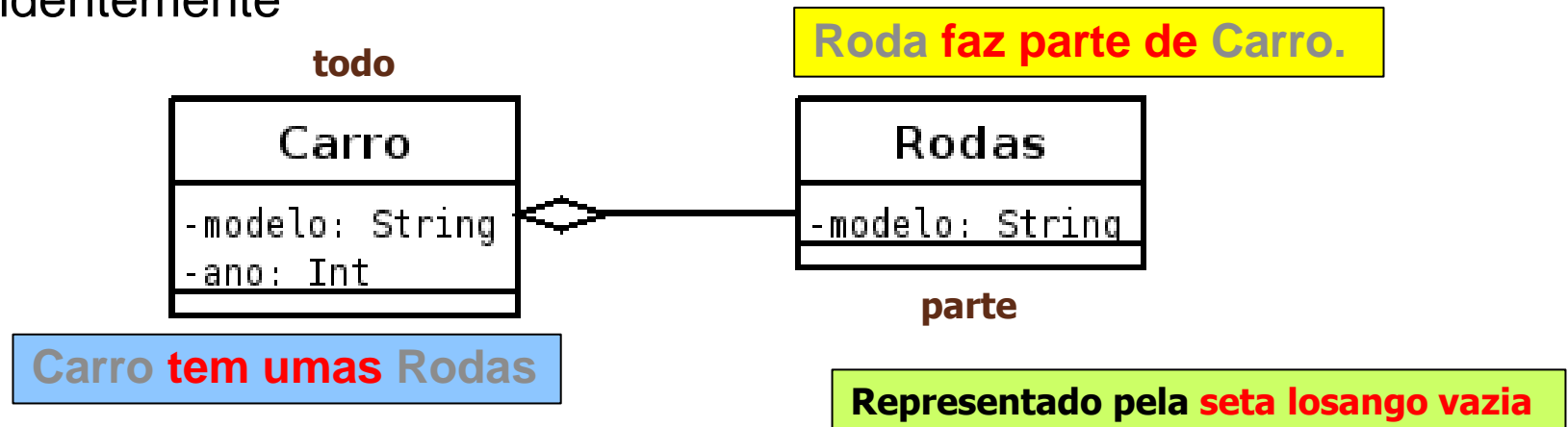
## Agregação

- A agregação é um caso particular da associação significando por exemplo que:
  - “B é parte de A”,
  - “A tem um B”,
  - “A é composto por B”
- A agregação estabelece uma relação *todo-parte* entre classes, sendo que *a parte pode existir sem o todo*.
- Embora as partes possam existir independentemente do todo, sua existência é basicamente para formar o todo



# Implementação de relações – Agregação

Ocorre quando o objeto é parte de outro objeto, mas eles podem existir independentemente



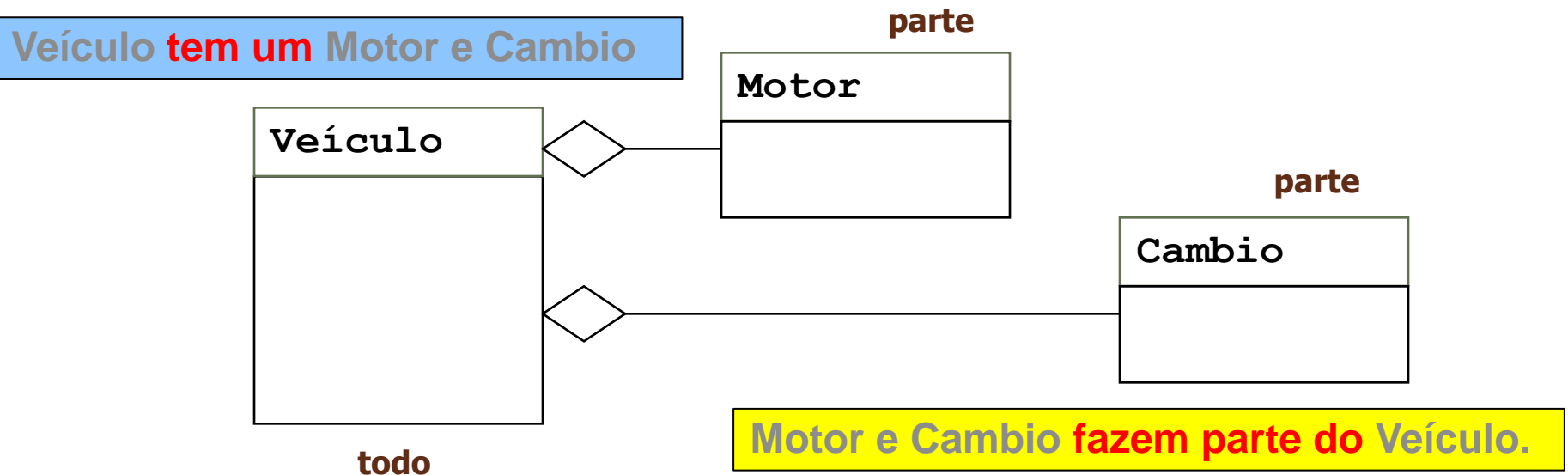
- Temos o objeto Carro que por sua vez faz referência ao objeto Rodas
- Porém o objeto Rodas pode existir mesmo que você destrua Carro
  - Você pode por exemplo remover a roda de um carro para colocar em outro.
- “Faz sentido a existência da Roda mesmo sem Carro e o Carro pode existir de preferência composto pelas Rodas”



# Implementação de relações – Agregação

## Outro Exemplo...

- A agregação estabelece uma relação *todo-parte* entre classes, sendo que *a parte pode existir sem o todo*.



- “Faz sentido a existência de Motor e Cambio mesmo sem o Veículo e o Veículo pode existir de preferência composto pelo Motor e Cambio”



# Implementação de relações – Agregação

## Agregação na perspectiva de implementação...

- Relações de 1 para 1 unidirecional



```
public class A {  
    ➡ private B b;  
    public A( ){  
    }  
    ➡ public void setB( B b  
    ){  
        this.b = b;  
    }  
    ➡ public B getB( ) {  
        return b;  
    }  
}  
  
public class B {  
    public B( ){  
    }  
}
```

**B "é parte de" A**

**A "tem um" B**

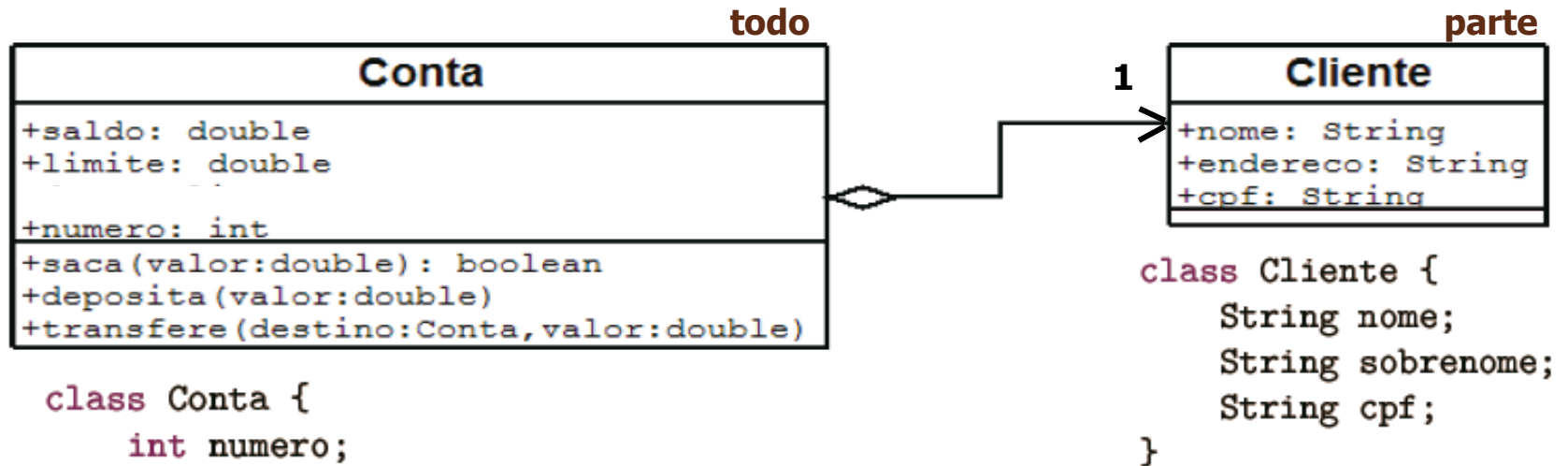
**De forma geral, uma agregação consiste de um objeto contendo referências para outros objetos, de tal forma que o primeiro seja o todo, e que os objetos referenciados sejam as partes do todo.**



# Implementação de relações – Agregação

## Agregação na perspectiva de implementação...

- Relações de 1 para 1 unidirecional



```
class Teste {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        Cliente c = new Cliente();  
        minhaConta.titular = c;  
        // ...  
    }  
}
```



# Implementação de relações – Composição

---

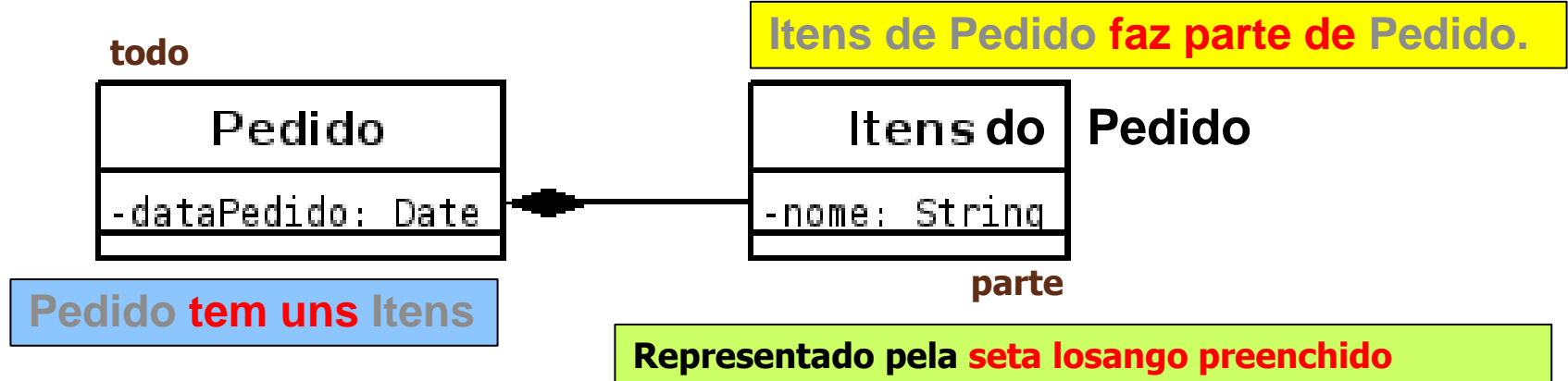
## Composição

- A composição é um caso particular da agregação significando por exemplo que:
  - “B é parte essencial de A”,
  - “A tem um B”,
  - “A é composto por B”
- A composição estabelece uma relação *todo-parte* entre classes, sendo que *a parte NÃO existe sem o todo*.
- A diferença é que o todo CONTÉM as partes (e não referências para as partes). Quando o todo desaparece, todas as partes também desaparecem.
  - As partes NÃO podem existir independentemente do todo, não faz sentido...



# Implementação de relações – Composição

Ocorre quando o objeto é parte de outro objeto, mas eles NÃO podem existir independentemente



- Temos o objeto Pedido que por sua vez faz referência ao objeto Itens,
- Portanto o objeto "Itens do Pedido" não faz sentido sem o objeto "Pedido".
- “Só faz sentido a existência da Itens de Pedidos se existir Pedido e Pedido e composto por Itens de Pedido”

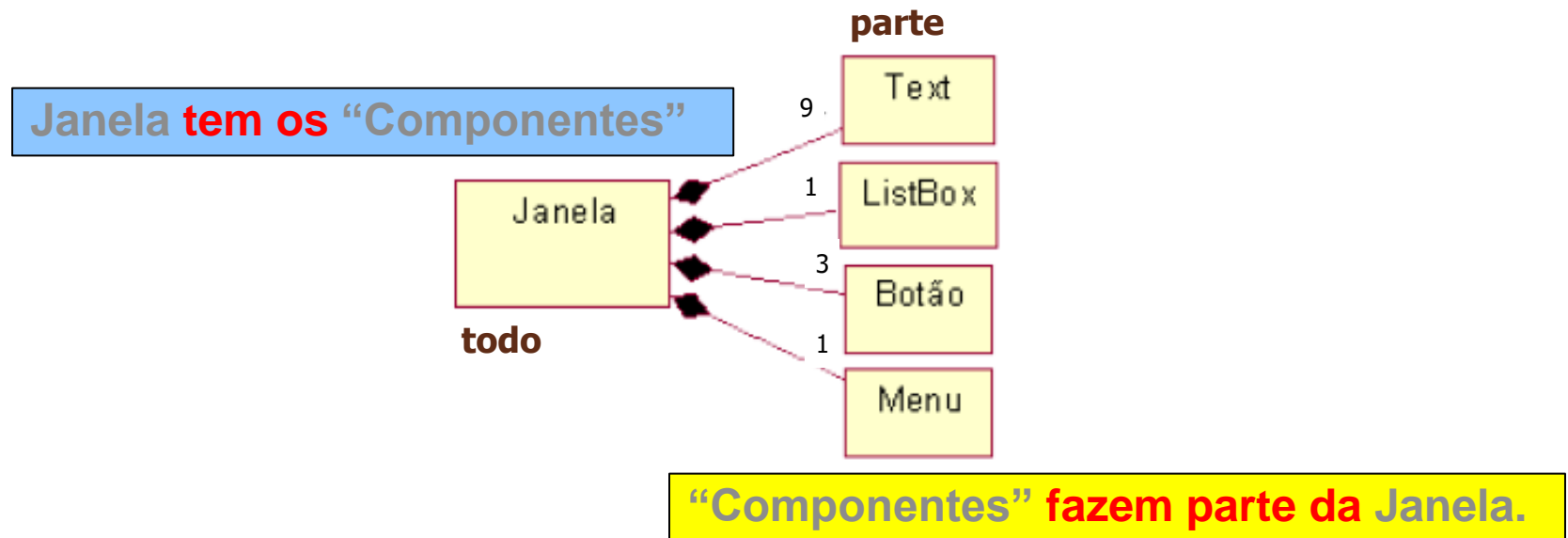




# Implementação de relações – Composição

## Outro Exemplo...

- A composição estabelece uma relação *todo-parte* entre classes, sendo que *a parte NÃO existe sem o todo*.



- “Não faz sentido a existência do Botão da Janela se a Janela não existir, já a Janela é composta pelos Componentes”



# Implementação de relações – Composição

## Composição na perspectiva de implementação...

- Relações de 1 para 1 unidirecional



```
public class A {
    ➡ private B b;
    public A() {
        ➡ b = new B();
    }
}

public class B {
    public B() {
    }
}
```

**B "é parte de" A**

**A "tem um" B**

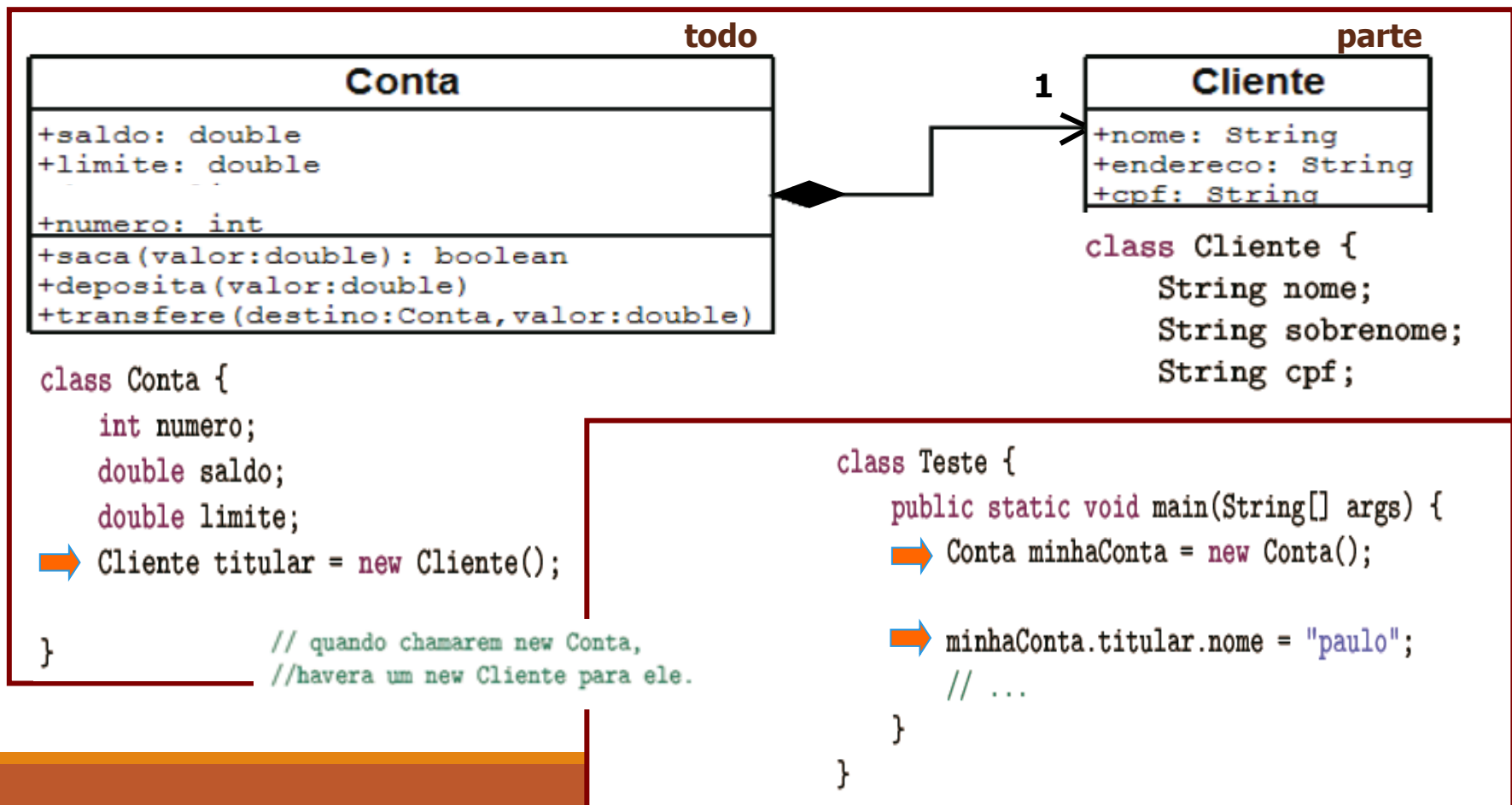
**De forma geral, uma composição consiste do todo contendo as partes (e não referências para as partes). Quando o todo desaparece, todas as partes também desaparecem.**



# Implementação de relações – Composição

## Agregação na perspectiva de implementação...

- Relações de 1 para 1 unidirecional





# Resumo Composição X Agregação X Associação

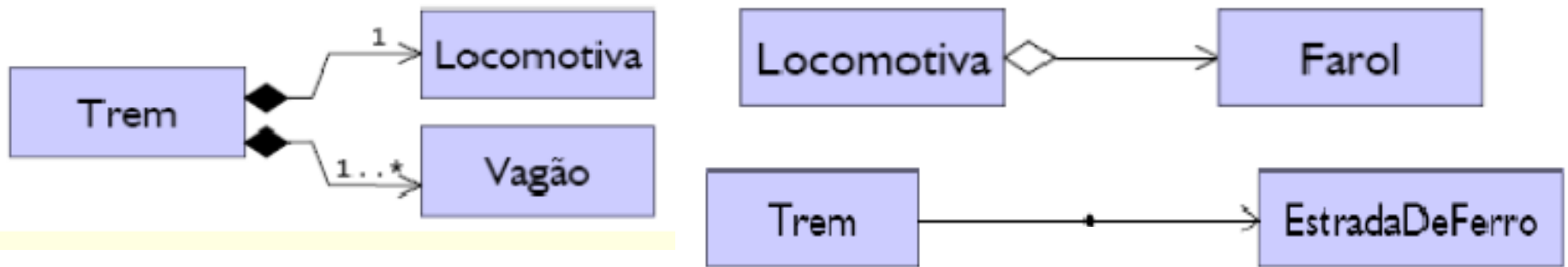
---

## De forma geral...

- Se NÃO HOUVER a relação todo-parte (excluindo os casos de herança e outras associações que não foram citadas)
  - -> ASSOCIAÇÃO SIMPLES
- Se HOUVER a relação todo-parte -> temos que ver se a parte pode existir sem o todo....
  - Se a parte existir sem o todo -> AGREGAÇÃO
  - Se a parte não existir sem o todo -> COMPOSIÇÃO



# Resumo Composição X Agregação X Associação



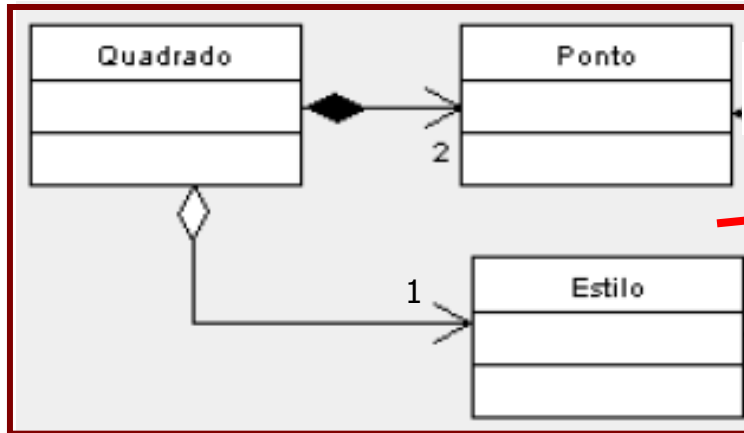
Um trem *não existe* sem a locomotiva e os vagões.

Uma locomotiva *possui* um farol (mas não vai deixar de ser uma locomotiva se não o tiver)

Um trem *usa* uma estrada de ferro (ela não faz parte do trem, mas ele depende dela)



# Resumo Composição X Agregação X Associação



```
public class Quadrado {
    // p1 e p2 são composição - new
    // estilo é agregação - atribuição
    ➡ private Ponto p1, p2;
    ➡ private Estilo estilo;
    public Quadrado(int x1, int y1,
                    int x2, int y2, Estilo e) {
        ➡ p1 = new Ponto(x1, y1);
        ➡ p2 = new Ponto (x2, y2);
        ➡ estilo = e;
    }
}
```



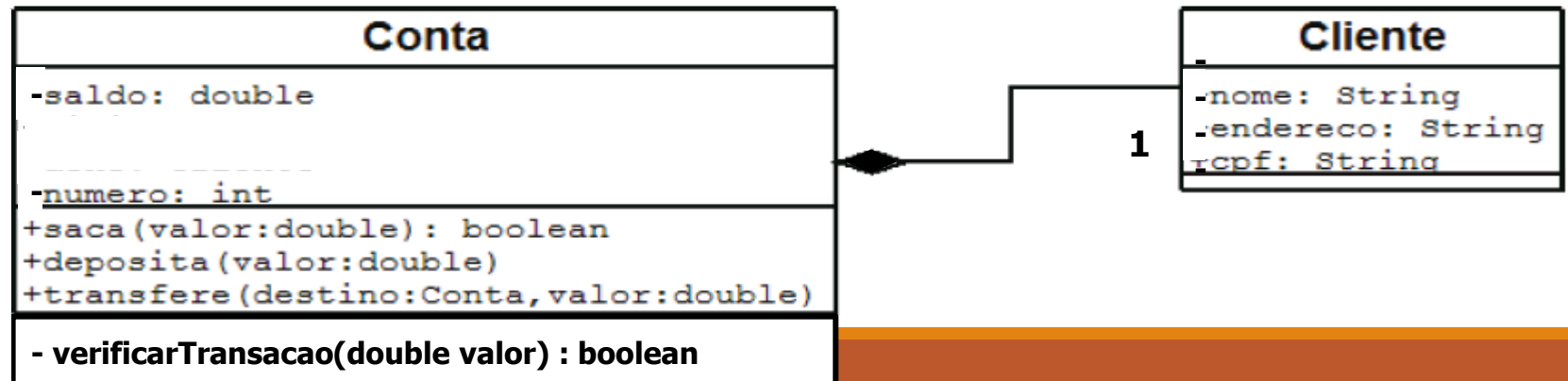
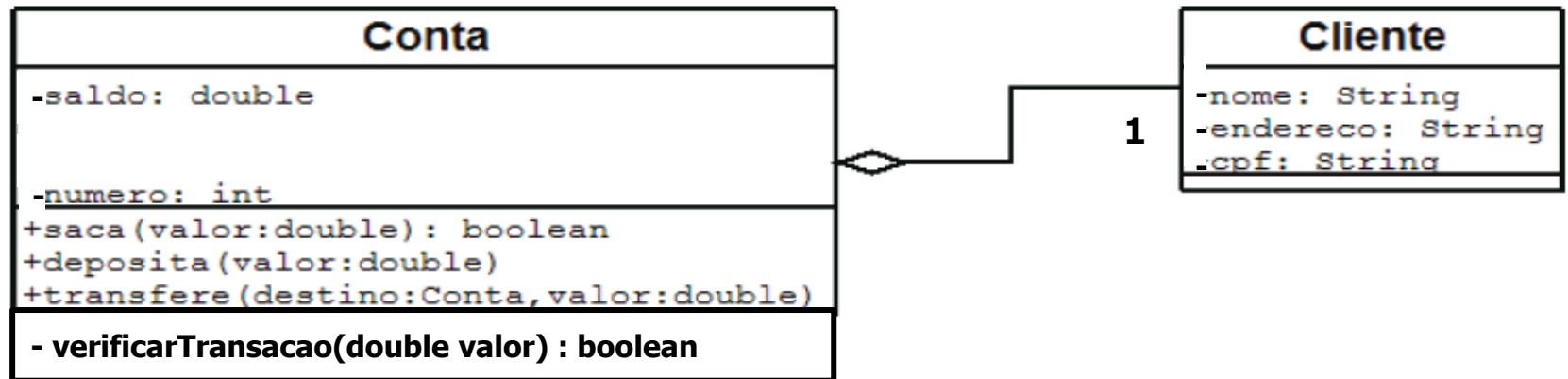
# Exercício Junto....

---

- Vamos agora construir juntos um exemplo que tenha:
  - Relações
  - Construtor (inicializando o saldo e o numero da Conta, por ex.) e default no cliente
  - Métodos acessores (atributos private)
  - this
- Baseado no seguinte diagrama....



# Exercício Junto....







# Exercício Junto....

## 1º. Como Agregação...

- Exemplo de main....

```
ClienteAgregacao cliente = new ClienteAgregacao();  
cliente.setNome("Vinicius");  
cliente.setCpf("123456789-10");  
cliente.setEndereco("Na sala de aula da UVV");
```

```
ContaAgregacao conta = new ContaAgregacao(01, 100);  
conta.setDono(cliente);
```

## 2º. Como Composição...

- Exemplo 

```
ContaComposicao conta = new ContaComposicao(01, 100);  
conta.getDono().setNome("Vinicius");  
conta.getDono().setCpf("123456789-10");  
conta.getDono().setEndereco("Na sala de aula da UVV");
```



UNIVERSIDADE  
**VILA VELHA**  
ESPÍRITO SANTO

# Exercícios

---

Agora é hora de exercitar.....

Tente resolver os seguintes problemas...

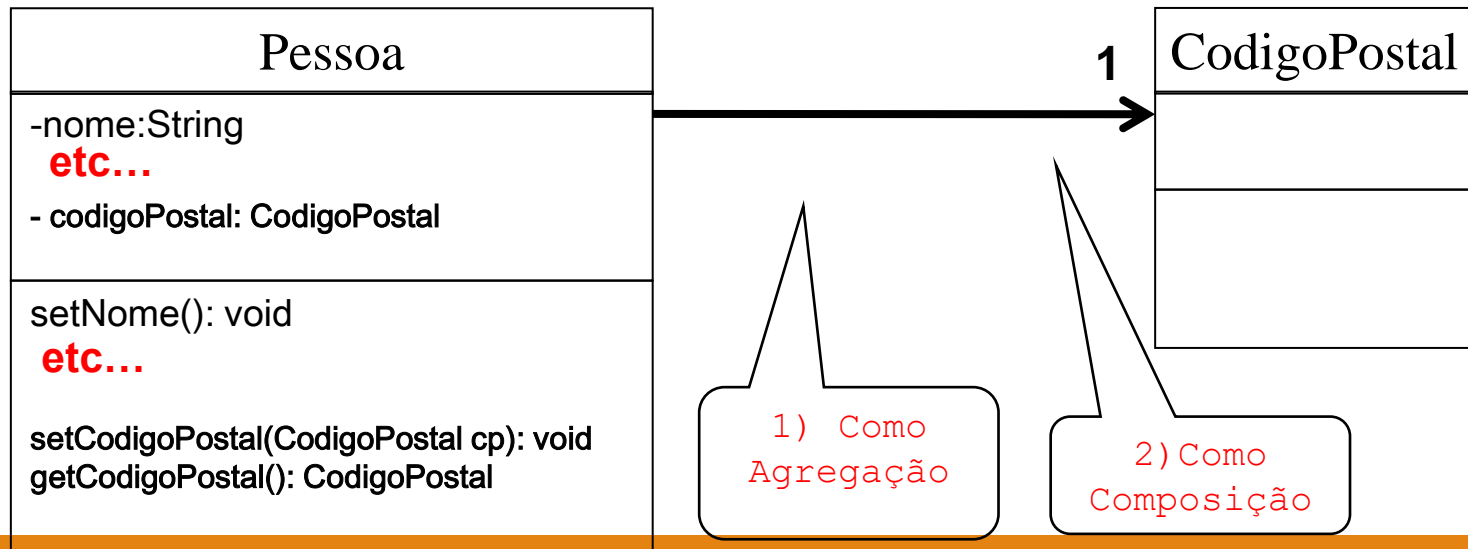
- Em dupla
- Apresentar ao professor no final da aula
- Pontuação em Atividades em sala de aula...
- Faça o JAVADOC de todos os exercícios!!!



# Exercício

## Relação unidirecional

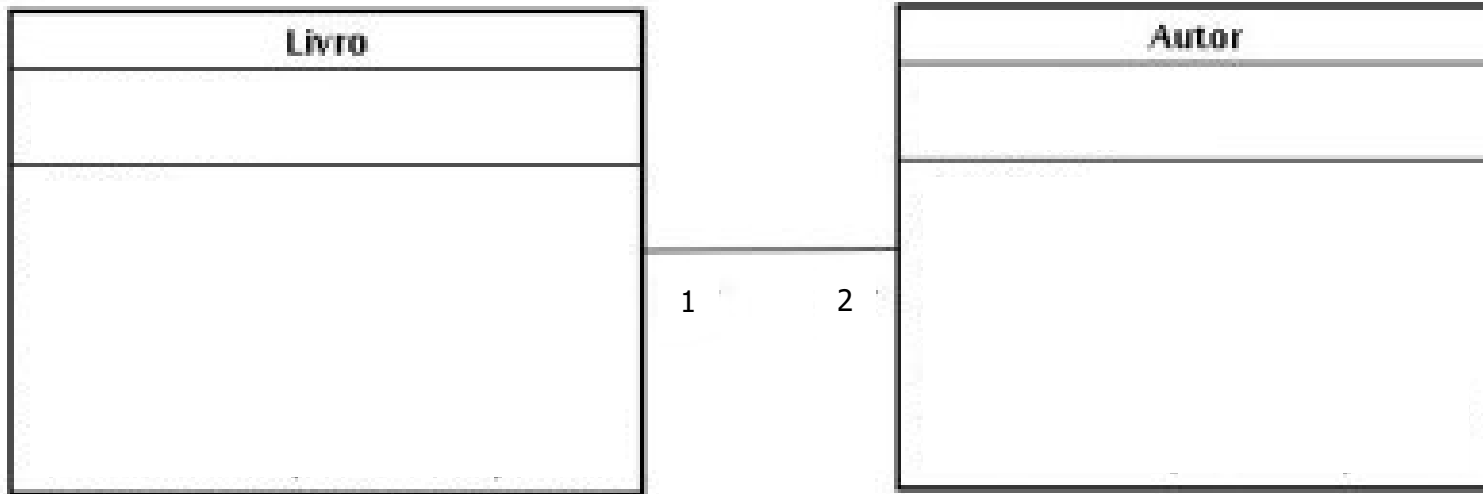
- Considere as classes Pessoa e CodigoPostal abaixo e implemente a relação existente entre ambas para cada caso indicado
- Cada instância de Pessoa possui um CodigoPostal.
- Escreva um programa de teste capaz de verificar a implementação da relação.





# Exercício

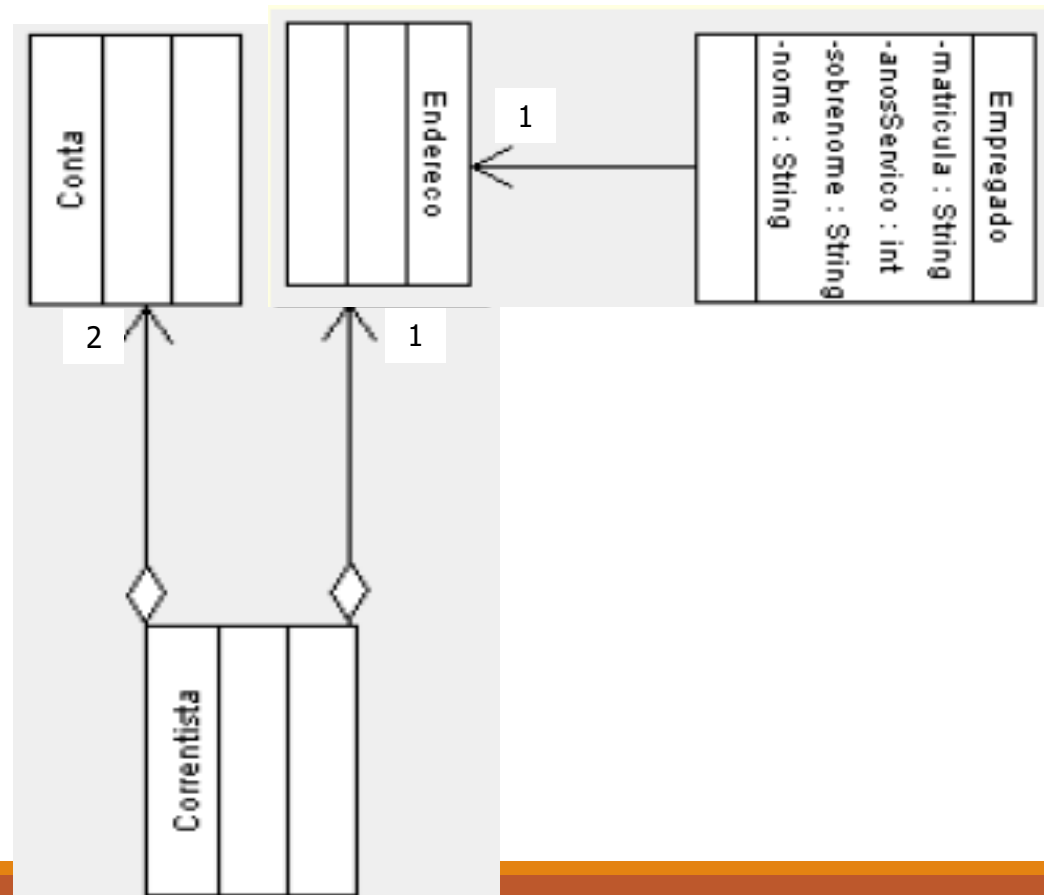
Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.





# Exercício

Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.





# Exercício

---

Faca uma classe Cartao que receba um objeto do tipo conta (exercício anterior) e uma senha.

- Ou seja, temos um relacionamento entre Conta e Cartao
- Um cartão pode ter até 3 contas associadas do mesmo cliente...

Devera conter um método retirada e um método saldo, semelhante à classe conta anterior,

- Mas que receba uma senha que devera ser a mesma armazenada no cartão.

—Faca também um método que altere a senha, desde que receba a senha antiga como parâmetro.



# Exercício

Crie uma classe Ponto que possua como atributos x e y do tipo double, e os seguintes métodos:

- Construtor,
  - Que deve receber por parâmetro os valores para inicializar os atributos e testar se os números são válidos, isto é, se estão dentro de limites pré-estabelecidos (entre 0 e 100);
- Especifique os métodos
  - void setX (double n), que permite alterar o valor do atributo x;
  - void setY(double n), que permite alterar o valor do atributo y;
  - double getX(), que retorna o valor do atributo x;
  - double getY(), que retorna o valor do atributo y.

Projete uma classe que possa utilizar a classe Ponto modelada.



# Exercício

Construa uma classe Quadrado que possua como atributos dois objetos do tipo Ponto (cantoSuperiorEsquerdo e cantoInferiorDireito) e os seguintes métodos:

- Construtor
  - Construtor, que deve receber duas instâncias da classe Ponto como parâmetro para inicializar os atributos;
- Especifique os métodos
  - void setCantoSuperiorEsquerdo(double x, double y), que permite alterar o valor do atributo cantoSuperiorEsquerdo;
  - void setCantoInferiorDireito(Ponto p), que permite alterar o valor do atributo cantoInferiorDireito;
  - Ponto getCantoSuperiorEsquerdo(), que retorna o valor do atributo cantoSuperiorEsquerdo;
  - Ponto getCantoInferiorDireito(), que retorna o valor do atributo cantoInferiorDireito;
  - String informa(), que retorna um objeto String que contém os valores dos atributos no formato "(x1,y1) - (x2,y2)".

Projete uma classe que possa utilizar a classe Quadrado modelada.