



C206/C06– Programação Orientada a Objetos  
com Java

# INTRODUÇÃO A ORIENTAÇÃO A OBJETOS

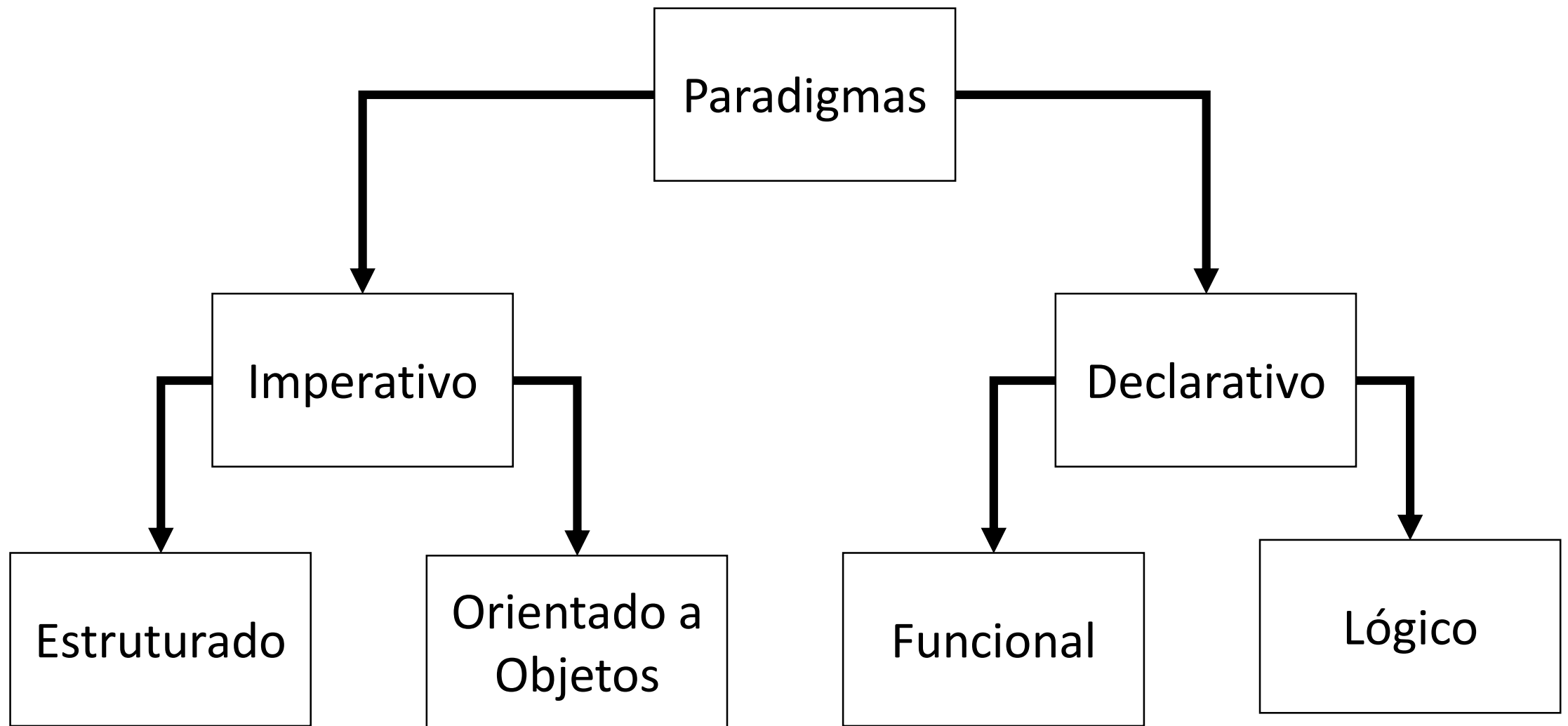
Prof. Christopher Lima  
christopher@inatel.br



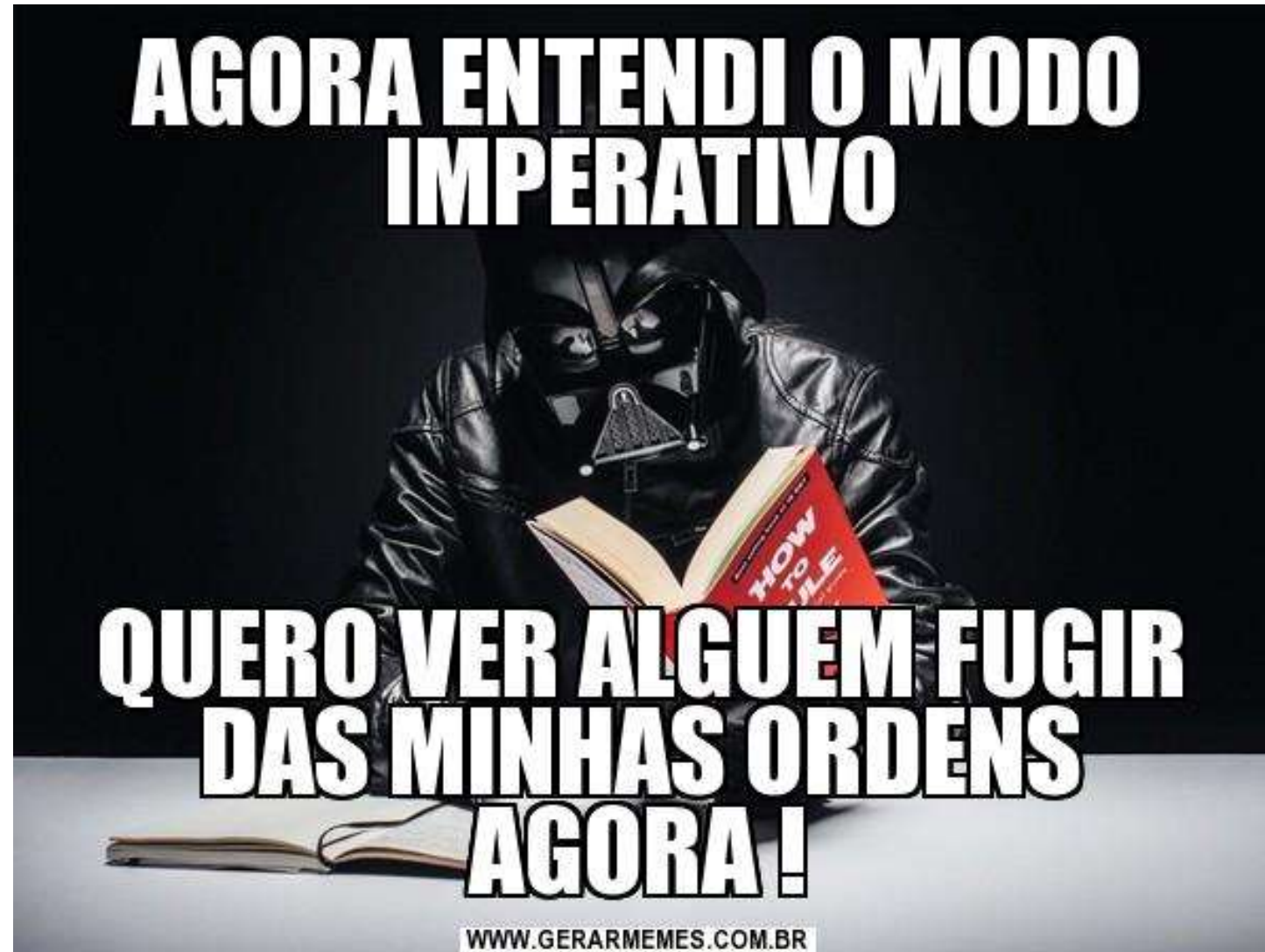
# Objetivos



- ☕ Comparativo entre programação estruturada e orientada a objetos (OO);
- ☕ Classes e Objetos;
- ☕ Uma classe em Java;
- ☕ Criando e usando um Objeto;
- ☕ Métodos;
- ☕ UML e Diagrama de Classes;
- ☕ Objetos são acessados por referências;
- ☕ Relacionamento entre objetos (Classes dentro de Classes);



# Imperativo



☕ Computação vista como um processo que realiza **mudança de estado**.

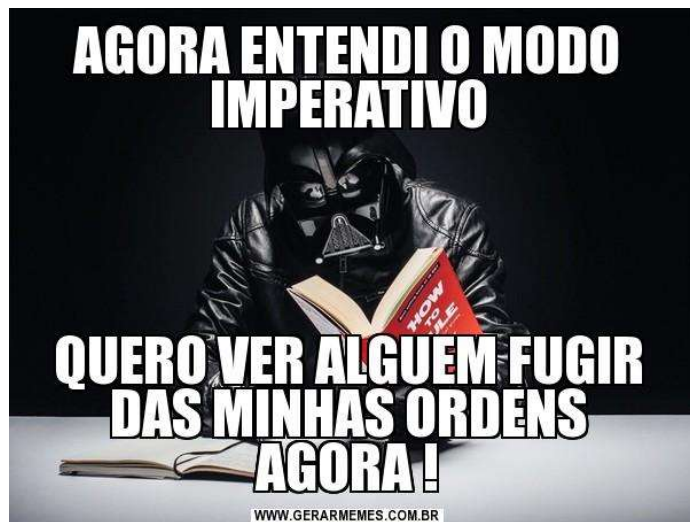
☕ Específica como um processamento **deve ser feito**

☕ Variáveis podem possuir diferentes **valores a cada momento (mudança de estado)**

```
int somar(int x, int y){  
    int z;  
    z = x + y;  
    return z;  
}
```

☕ Faça a soma de X com Y

☕ Me devolva o resultado



Estruturado





☕ Se baseia na construção de blocos aninhados de comandos.





☕ Utiliza três mecanismos básicos:

☕ Sequencia

☕ Seleção

☕ Iteração

☕ Exemplo de linguagem: C e Pascal

# Sequencia

```
int x;
```

```
int y;
```

```
int z;
```

```
x = x * 10;
```

```
y = x + z;
```

```
trocar(x,y);
```



# Seleção



```
if (x > 10){  
    printf("x maior que 10");  
}  
else{  
    printf("x não é maior que 10");  
}
```

# ☕ Iteração



```
for (int i = 0; i < 10; i++){  
    printf("Imprimindo o número: %d", i);  
}
```

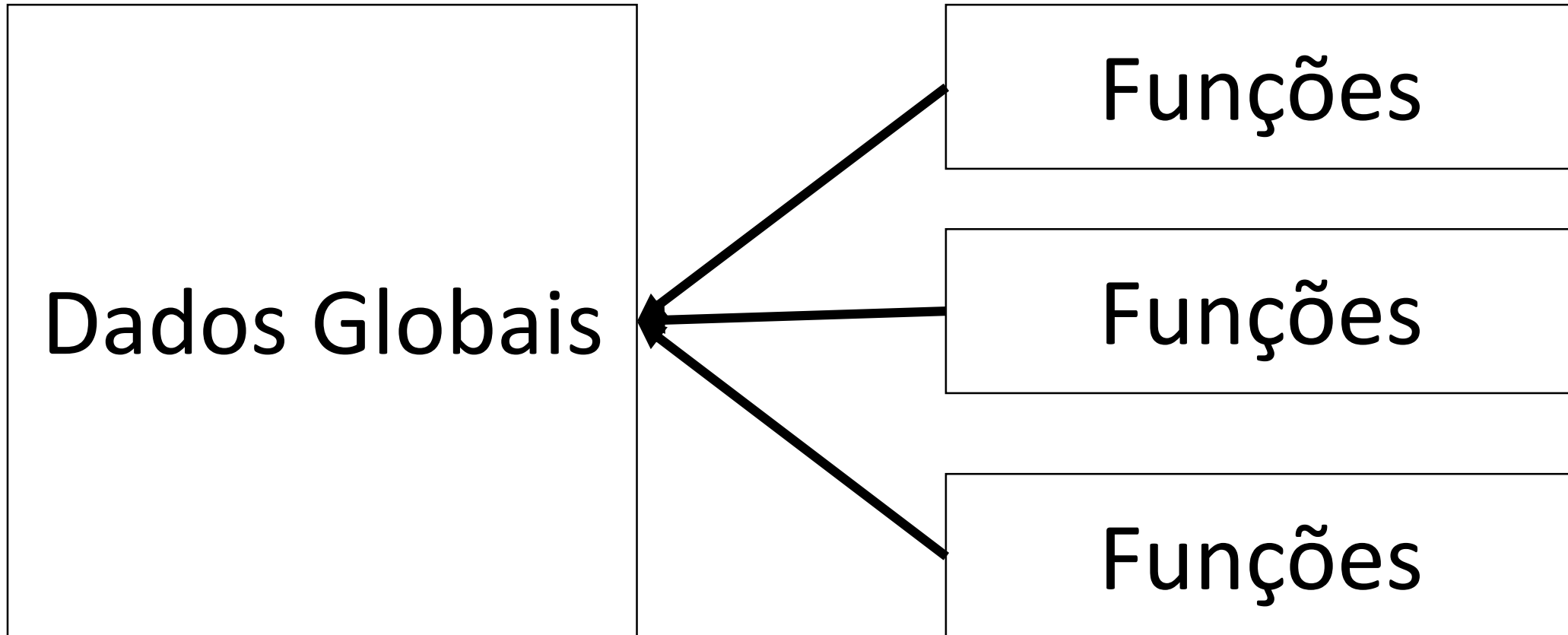


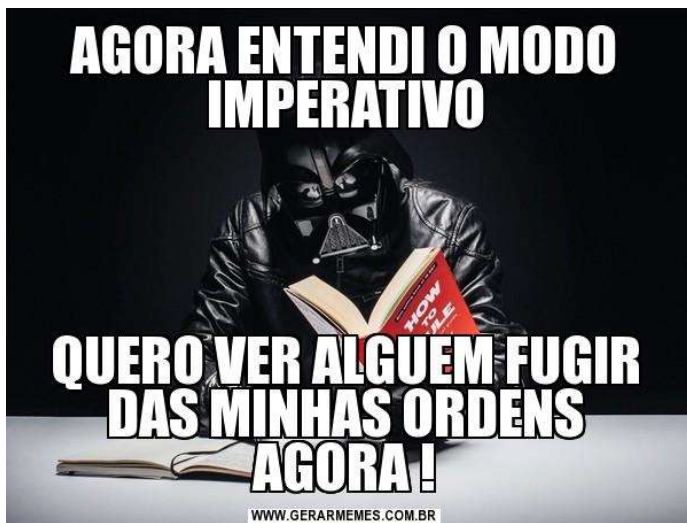
☕ E os dados?

☕ Baseado em **variáveis globais**

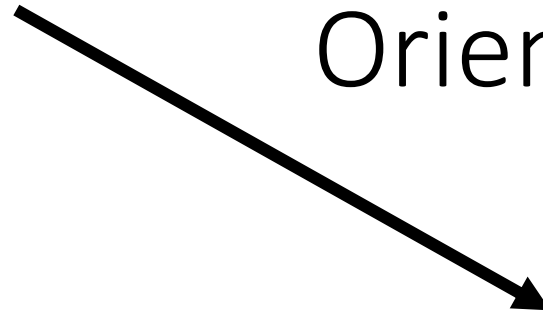
☕ Podem ser **acessados de qualquer parte do código sem a necessidade de permissão**

# ☕ E os dados?





Orientado a Objetos

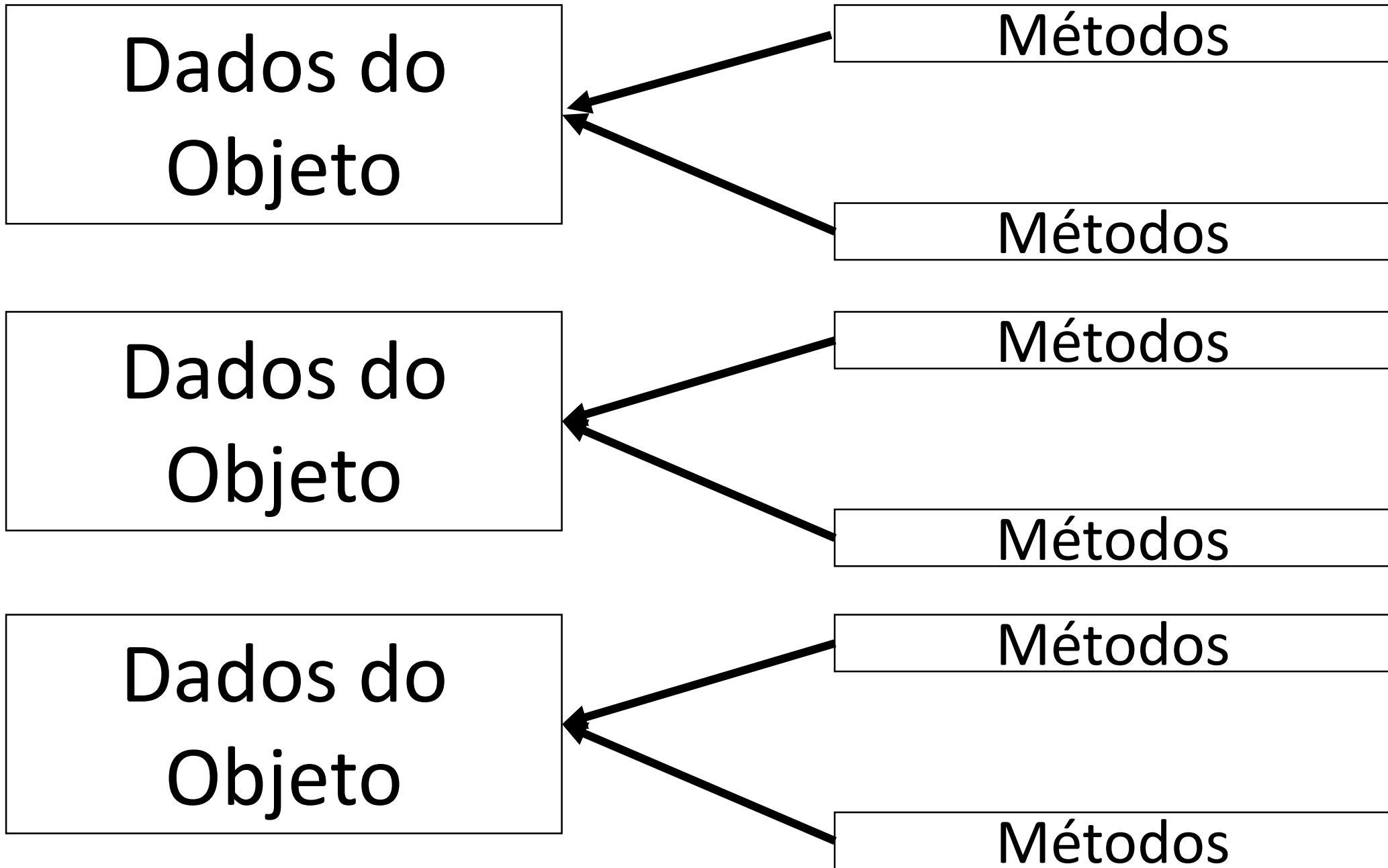


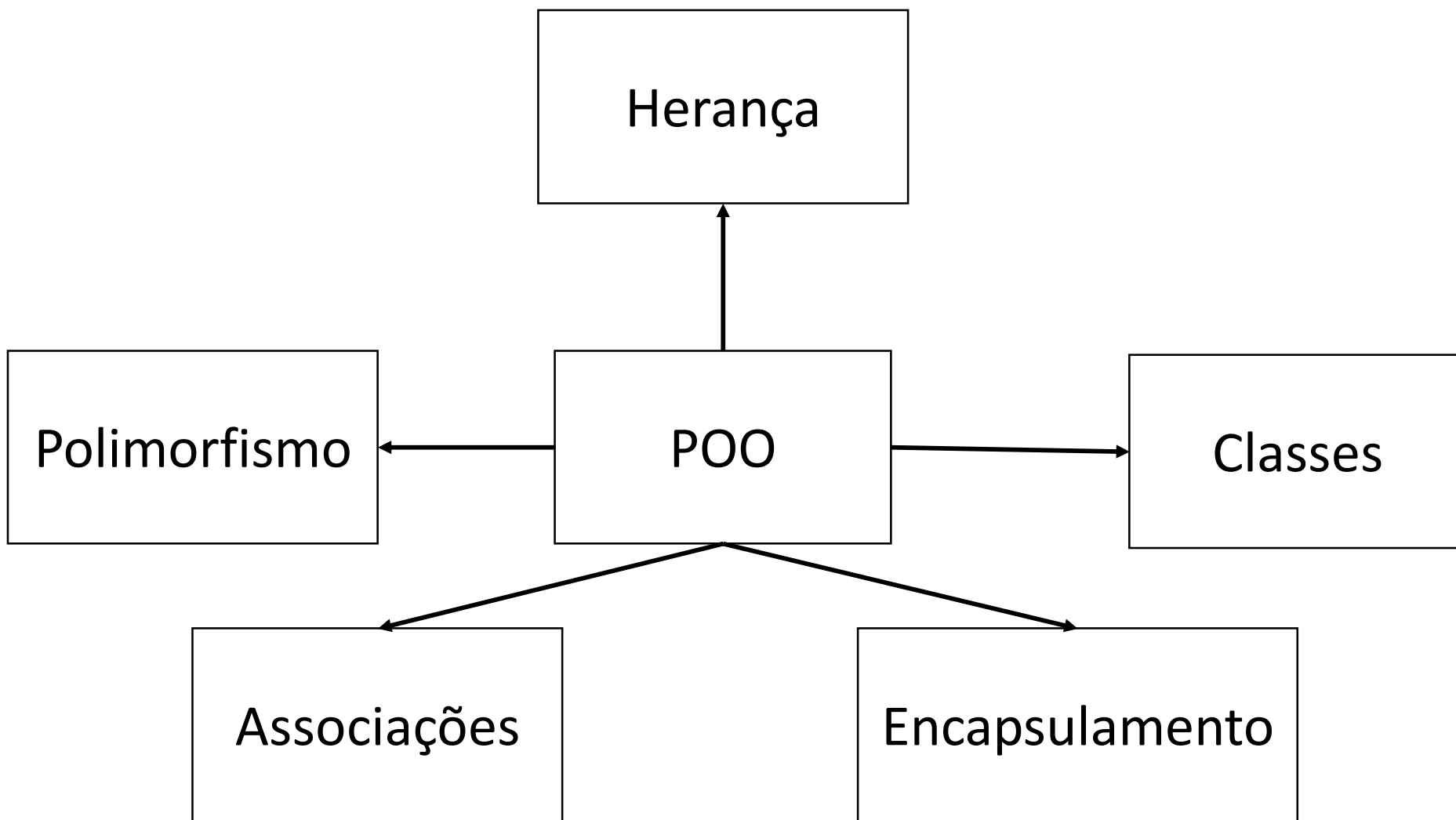
☕ Visto como uma **evolução** do paradigma estruturado;

☕ Apresenta **melhor organização dos dados**

☕ Exemplo: Java, C#, C++







# Exemplo de Classe em Java

```
public class Pessoa{
```

```
    String nome;
```

```
    int idade;
```

```
    public void falar(){
```

```
        System.out.println("Olá alunos(as) de C06/C206");
```

```
    }
```

```
}
```

# Exemplo de Herança em Java

```
public class Professor extends Pessoa{  
    String nome;  
    public void ministrarAula(){  
        //ministra aula de POO  
    }  
}
```

# Exemplo de Associação em Java

```
public class Faculdade{  
    String nome;  
    Professor professor; //Associação  
    public void ministrarAulas(){  
        professor.ministrarAula(); //Quem ministra  
aula é o professor  
    }  
}
```

# Exemplo de Encapsulamento

```
public class Faculdade{  
    public String nome;  
    private Professor professor; //Associação  
    public void ministrarAulas(){  
        professor.ministrarAula(); //Quem ministra  
aula é o professor  
    }  
}
```

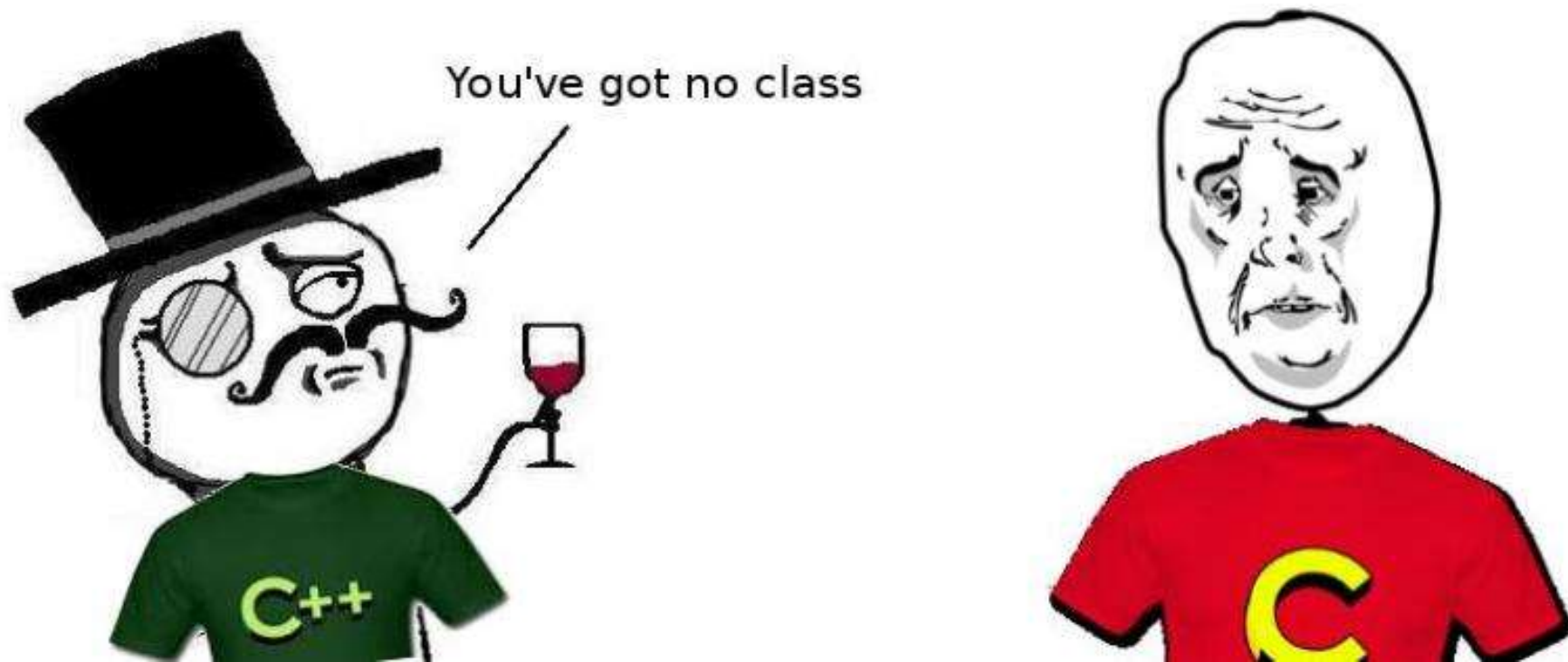


☕ Por que os cursos de graduação se iniciam com programação estruturada?

☕ Por que é muito mais simples iniciar um código estruturado! 😊



# Classes e Objetos





☕ Vamos criar um programa para um banco

☕ Uma abstração que precisaremos é uma  
CONTA!

☕ Precisamos responder

☕ O que uma CONTA possui?

☕ O que uma CONTA faz?



☕ O que uma CONTA possui? (Membros da classe)

☕ Saldo;

☕ Limite;

☕ Número da Conta;

☕ Dono da Conta;

☕ O que uma CONTA sabe fazer? (Métodos)

☕ Sacar;

☕ Depositar;

☕ Transferir

☕ Verificar saldo;

☕ Essa conta é apenas uma especificação!

☕ Ainda não podemos utilizá-la

☕ Para utilizar uma conta, precisamos construí-la e obter uma **INSTÂNCIA** da classe!

☕ A classe é uma especificação

☕ A instância é o objeto concreto





## Especificação

número:  
cliente:  
saldo:  
limite:

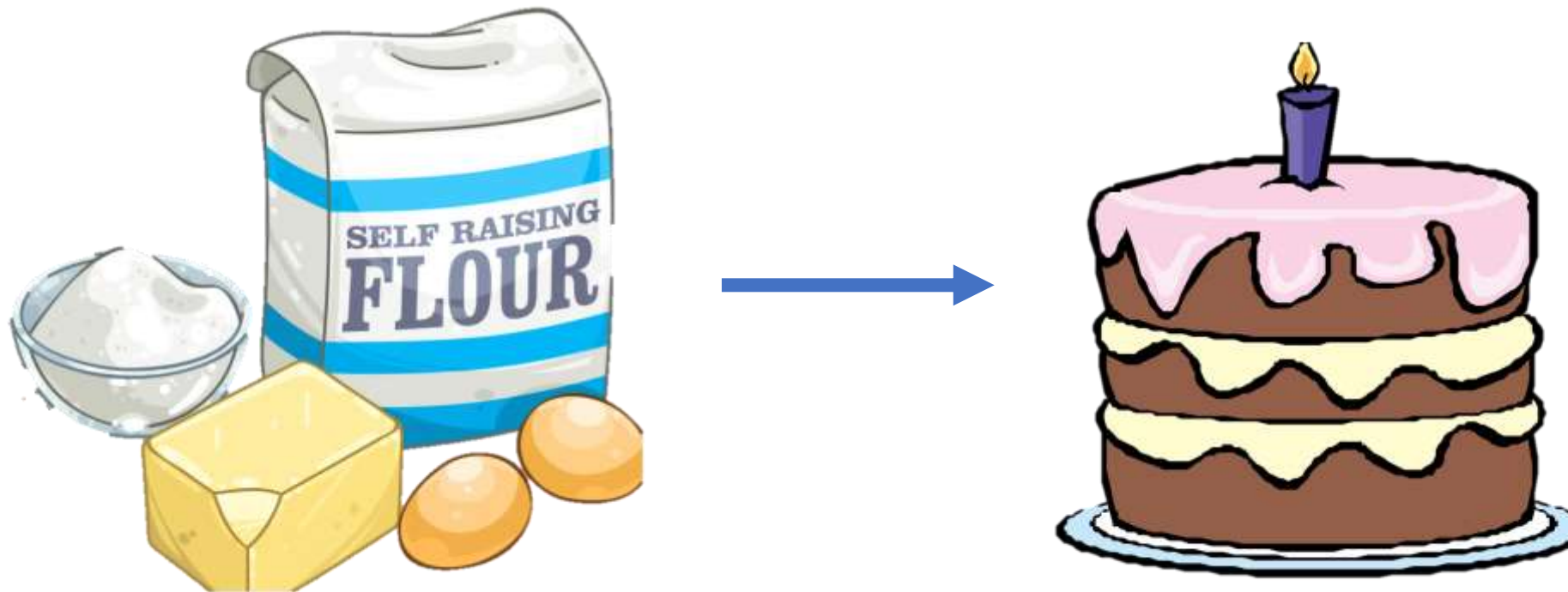
## Instância 1

número: 1234  
cliente: Maria  
saldo: 1000,00  
limite: 2500,00

## Instância 2

número: 4321  
cliente: João  
saldo: 700,00  
limite: 1300,00

- ☕ Na vida não comemos a receita!
- ☕ Precisamos "construir" o bolo primeiro



# Vamos criar a nossa classe Conta



```
public class Conta {
```

```
    //Membros da Classe
```

```
    int numero;
```


```
    String nomeDoDono;
```

```
    float saldo;
```

```
    float limite;
```

```
    //Metodos
```

```
}
```

 Variáveis declaradas no escopo da classe chamamos de "Membros da classe" (ou "atributos")

 Campos da classe (*field*)

☕ Ok! Temos nossa especificação de Conta, mas como usá-la?



☕ Precisamos de uma instância de Conta

```
public class Main {  
    public static void main(String[] args) {  
        //Criando uma nova instancia de Conta!  
        new Conta();  
    }  
}
```

☕ Mas onde ela está sendo salva?

☕ Para o além!

☕ Precisamos atribuir a instância de Conta para alguma variável!



☕ Ela precisa ser capaz de salvar um dado tipo "Conta".

```
public class Main {  
  
    public static void main(String[] args) {  
  
        //Criando uma nova instancia de Conta!  
        //E atribuindo a uma variável do tipo Conta!  
        Conta c = new Conta();  
  
    }  
  
}
```



# Atribuindo valores e acessando esses valores!



```
public static void main(String[] args) {  
  
    Conta c = new Conta();  
    c.nomeDoDono = "Joaquina";  
    c.saldo = 1000;  
  
    System.out.println("O dono da classe eh: " + c.nomeDoDono  
        + " e o saldo eh: " + c.saldo);  
}
```

☕ E as ações (comportamento) da classe Conta?

☕ Vamos criar métodos para isso.

☕ Como podemos sacar e depositar nessa conta?

☕ Fazemos isso na classe Conta!



```
//Metodo para depositar
void deposita(float quantia) {

    saldo += quantia;
}

//Metodo para sacar
void saca(float quantia) {

    float novoSaldo = saldo - quantia;
    saldo = novoSaldo;
}
```



☕ Invocando o método!

☕ O operador "." tem essa função!

```
//Depositando R$ 100,00  
c.deposita(100);  
System.out.println(c.saldo);
```

```
//Sacando R$ 50,00  
c.saca(50);  
System.out.println(c.saldo);
```



# Exercício 1

## Exercício 1 – Zumbis





# Exercício 1

## Exercício 1 – Zumbis

Você e seus amigos querem criar um software para modelar zumbis. Crie uma classe que representa um zumbi. O que um zumbi sabe sobre si? E o que ele sabe fazer?

Depois crie uma classe Principal (App/Main) que use o zumbi e invoque suas ações.

As ações podem ser mostradas via `System.out.println();`

# Objetos são acessados por referencia!



☕ Quando declaramos uma variável para guardar um objeto, estamos na verdade guardando uma REFERÊNCIA para esse objeto.

☕ Exemplo:

☕ `Conta c = new Conta();`

☕ “c” é um objeto.

☕ “c” também é uma variável que guarda uma REFERÊNCIA para uma INSTÂNCIA de Conta



# Objetos são acessados por referencia!

☕ As variáveis `c1` e `c2` são “ponteiros” para o objeto `Conta`

☕ Mas não podemos fazer a aritmética de ponteiros com essas variáveis!!!!!!

```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

# Objetos são acessados por referencia!



☕ Podemos comparar variáveis de referências?

☕ Sim, mas fique atento!





# Objetos são acessados por referencia!



```
Conta c1 = new Conta();  
c1.nomeDoDono = "Jaum";  
c1.limite = 1000;  
c1.numero = 1234;  
c1.saldo = 2000;
```

```
Conta c2 = new Conta();  
c2.nomeDoDono = "Jaum";  
c2.limite = 1000;  
c2.numero = 1234;  
c2.saldo = 2000;
```

Não são iguais!

```
if(c1 == c2)  
    System.out.println("São iguais!");  
else  
    System.out.println("Não são iguais!");
```



# Objetos são acessados por referencia!



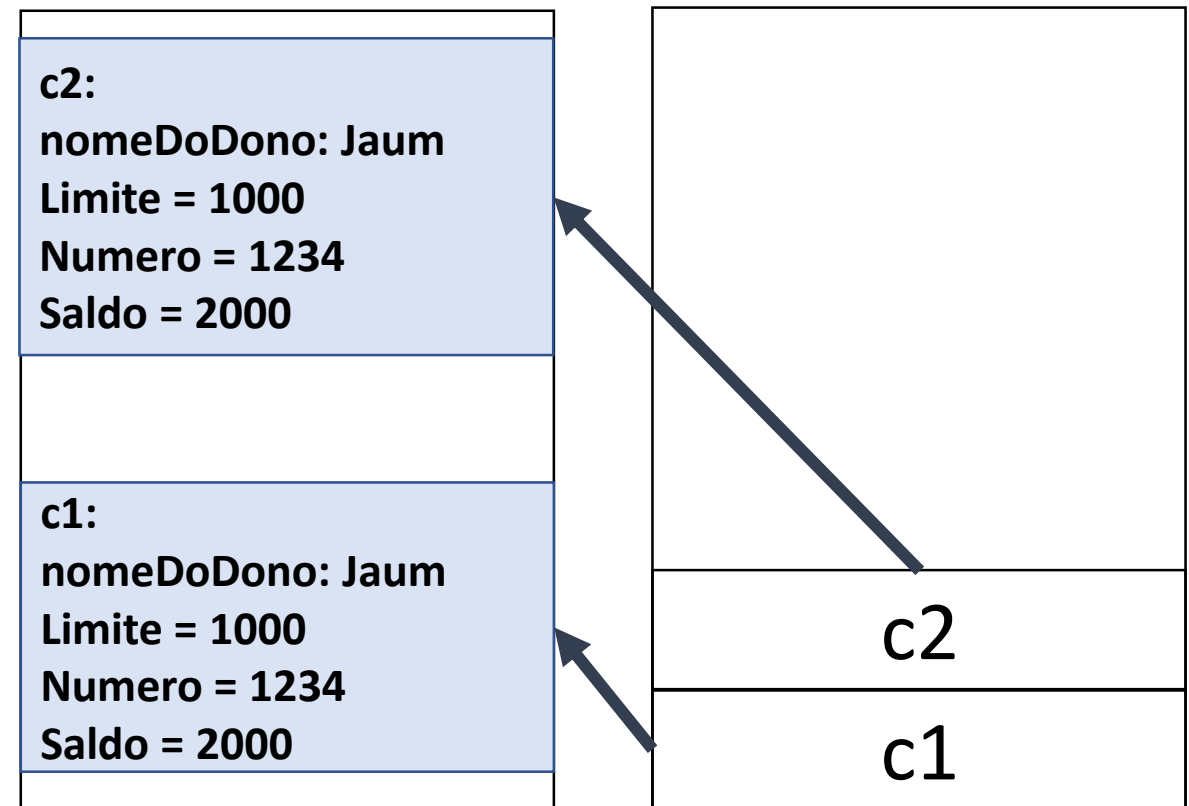
```
Conta c1 = new Conta();  
c1.nomeDoDono = "Jaum";  
c1.limite = 1000;  
c1.numero = 1234;  
c1.saldo = 2000;
```

```
Conta c2 = new Conta();  
c2.nomeDoDono = "Jaum";  
c2.limite = 1000;  
c2.numero = 1234;  
c2.saldo = 2000;
```

```
if(c1 == c2)  
    System.out.println("São iguais!");  
else  
    System.out.println("Não são iguais!");
```

Heap

Stack



# Objetos são acessados por referencia!



## São iguais!

```
Conta c1 = new Conta();  
c1.nomeDoDono = "Jaum";  
c1.limite = 1000;  
c1.numero = 1234;  
c1.saldo = 2000;
```

```
Conta c2 = new Conta();  
c2.nomeDoDono = "Jaum";  
c2.limite = 1000;  
c2.numero = 1234;  
c2.saldo = 2000;
```

```
c1 = c2;
```

```
if(c1 == c2)  
    System.out.println("São iguais!");  
else  
    System.out.println("Não são iguais!");
```



# Objetos são acessados por referência!



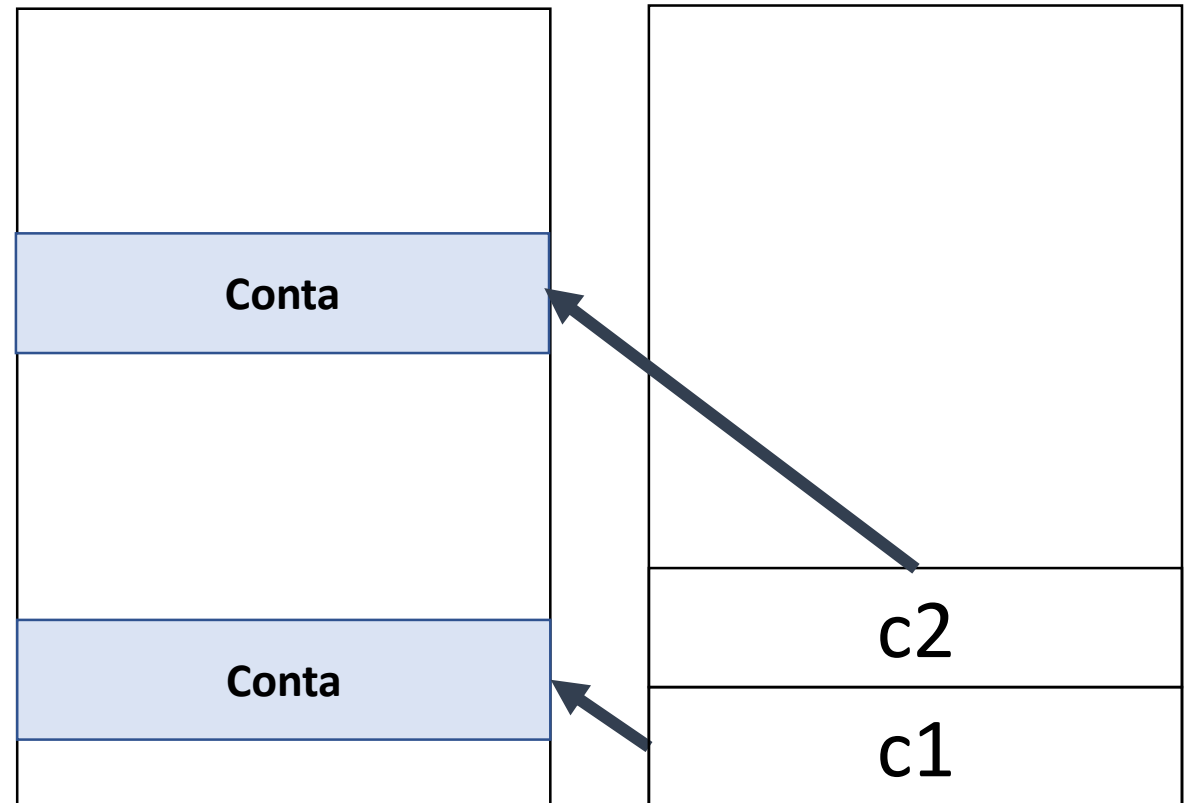
```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

```
c1 = c2;
```

Heap

Stack



# Objetos são acessados por referencia!



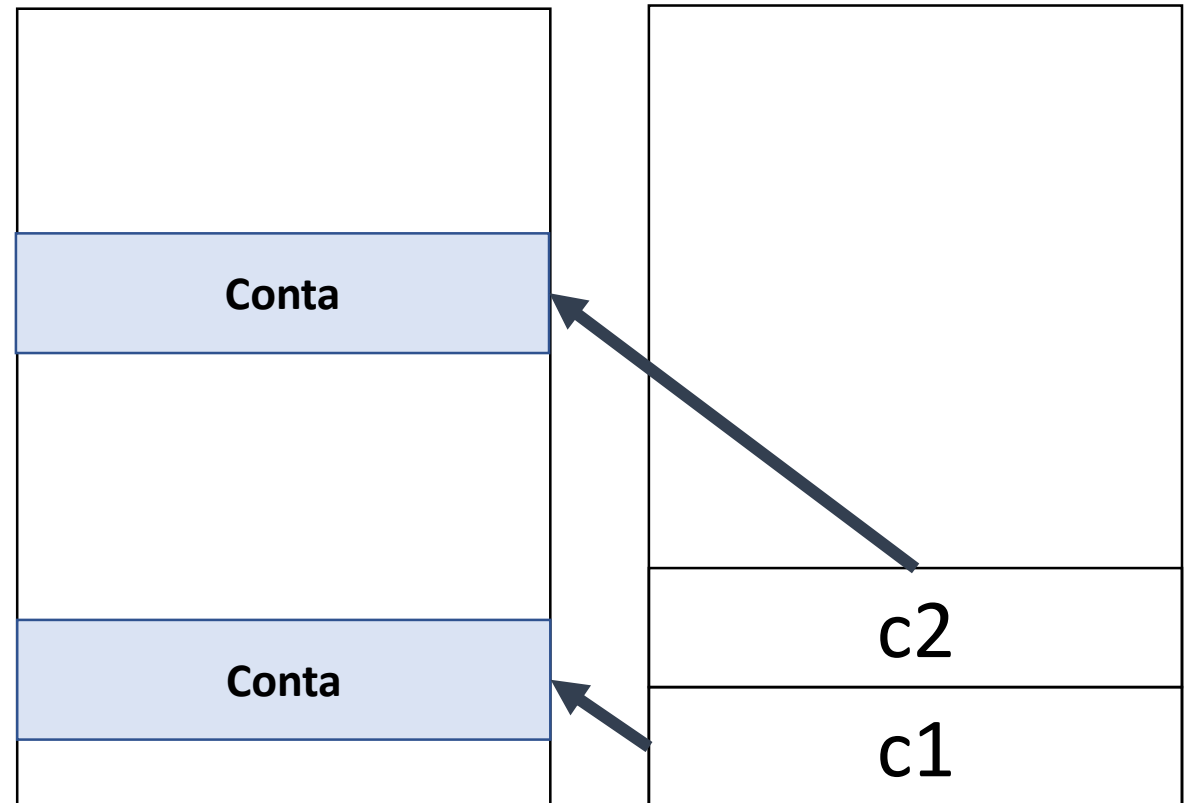
```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

```
c1 = c2;
```

Heap

Stack



# Objetos são acessados por referencia!



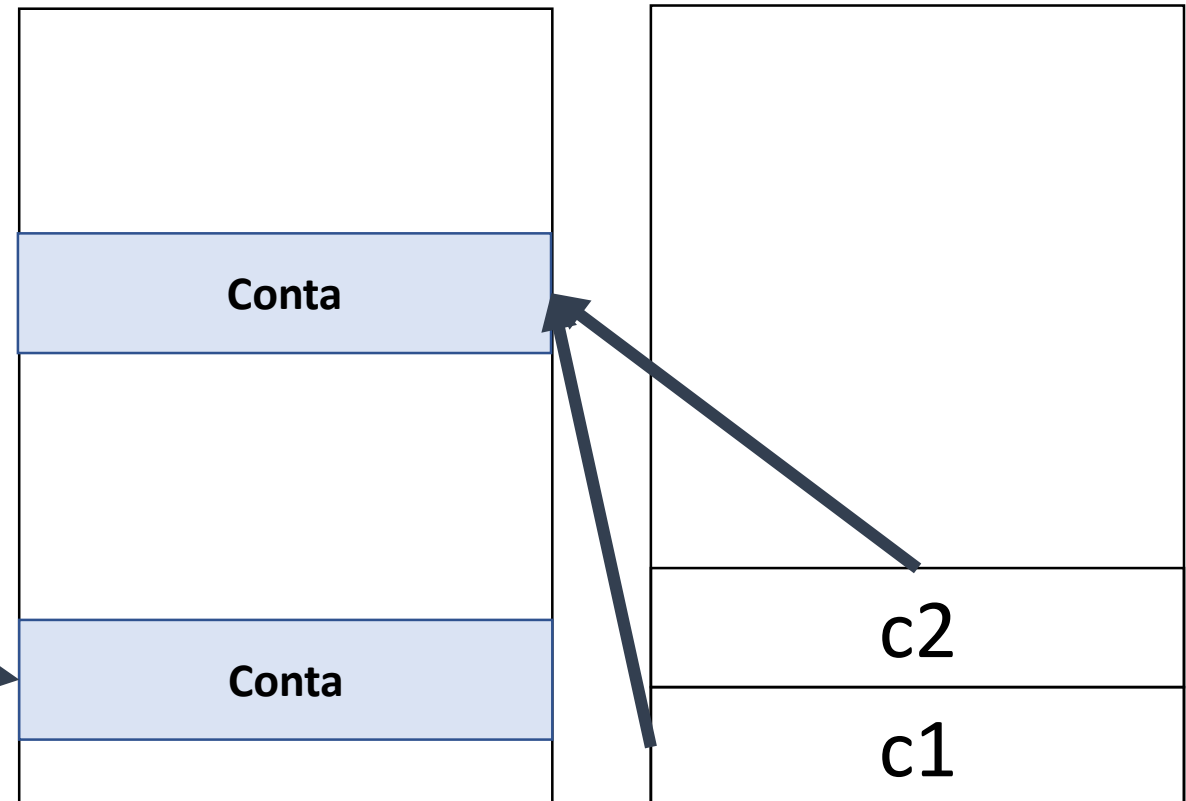
```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

```
c1 = c2;
```

Heap

Stack



Garbage Collector





# Vamos transferir dinheiro!!!!

☕ Na classe Conta, criamos o método “transferir”.

☕ Ele recebe como parâmetro, outra Conta!

```
void transferir(Conta contaDestino, float quantia) {  
    saldo -= quantia;  
    contaDestino.saldo += quantia;  
}
```



# Vamos transferir dinheiro!!!!

☕ Vamos testar no método “main”

```
c1.transferir(c2, 100);  
System.out.println("O saldo de c1 é: " + c1.saldo);  
System.out.println("O saldo de c2 é: " + c2.saldo);
```

```
O saldo de c1 é: 1900.0
```

```
O saldo de c2 é: 2100.0
```

☕ Lembre-se: Objetos são passados por referência!



# UML (Unified Modeling Language)!



- ☕ É uma linguagem visual para especificação, visualização e documentação de software!
- ☕ O diagrama de classes, é um dos vários tipos de diagramas oferecidos pela linguagem UML
- ☕ Considere o diagrama UML da classe Conta

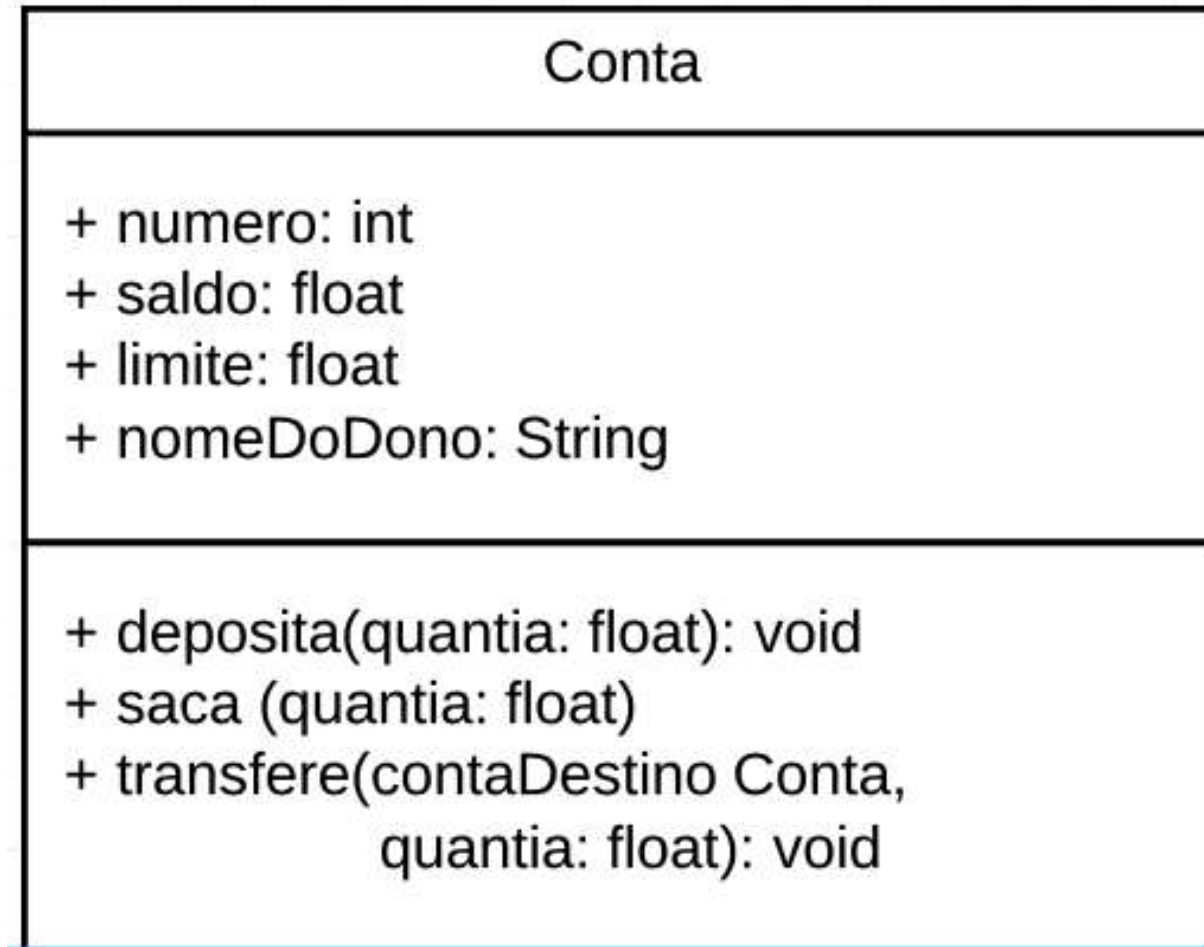


# UML (Unified Modeling Language)!

Nome da Classe →

Membros da Classe →

Métodos →

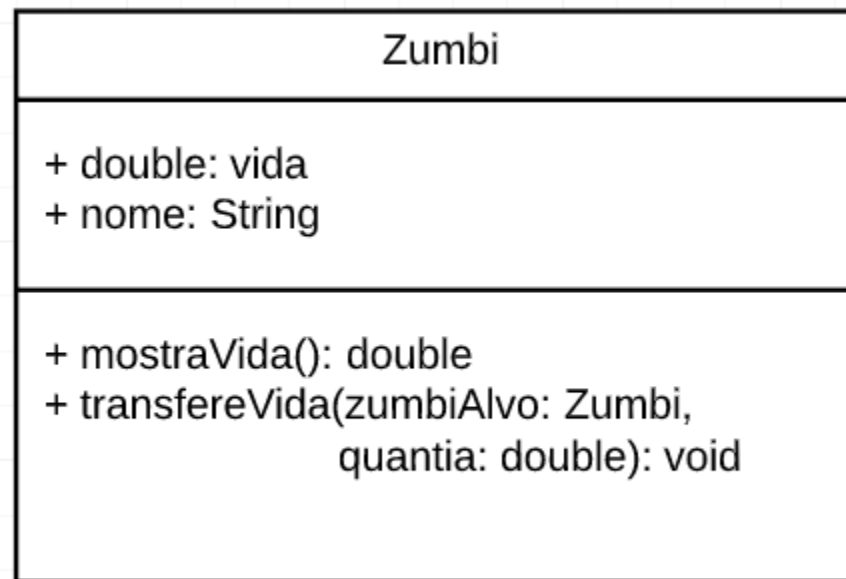




## Exercício 2

### Exercício 2 – Zumbis!

Você e seus amigos **AINDA** estão criando um software para modelar zumbis! Considere o UML para modelar o zumbi!





## Exercício 3

### Exercício 3 – Zumbis!

Você e seus amigos **CONTINUAM** criando um software para modelar zumbis! Considerando o zumbi do exercício 2

- Crie dois zumbis (cada um com seu “new”).
- Manipule a vida individualmente
- Depois faça ambas as variáveis de referencias iguais
  - Ex: `z1 = z2;`
- Manipule a vida dos zumbis (através de `z1` e `z2`).
- Verifique a vida deles



## Exercício 4

### Exercício 4 – Zumbis!

Sim isso mesmo, ainda estão no zumbis! Modifique o Zumbi do Exercício 2 para atender a seguinte especificação

- A ação de transferir vida faz uma verificação antes se de fato é possível ser realizada.
- Ele retorna um booleano informando se deu certo a transferência
- Modifique o Diagrama para atender o novo requisito





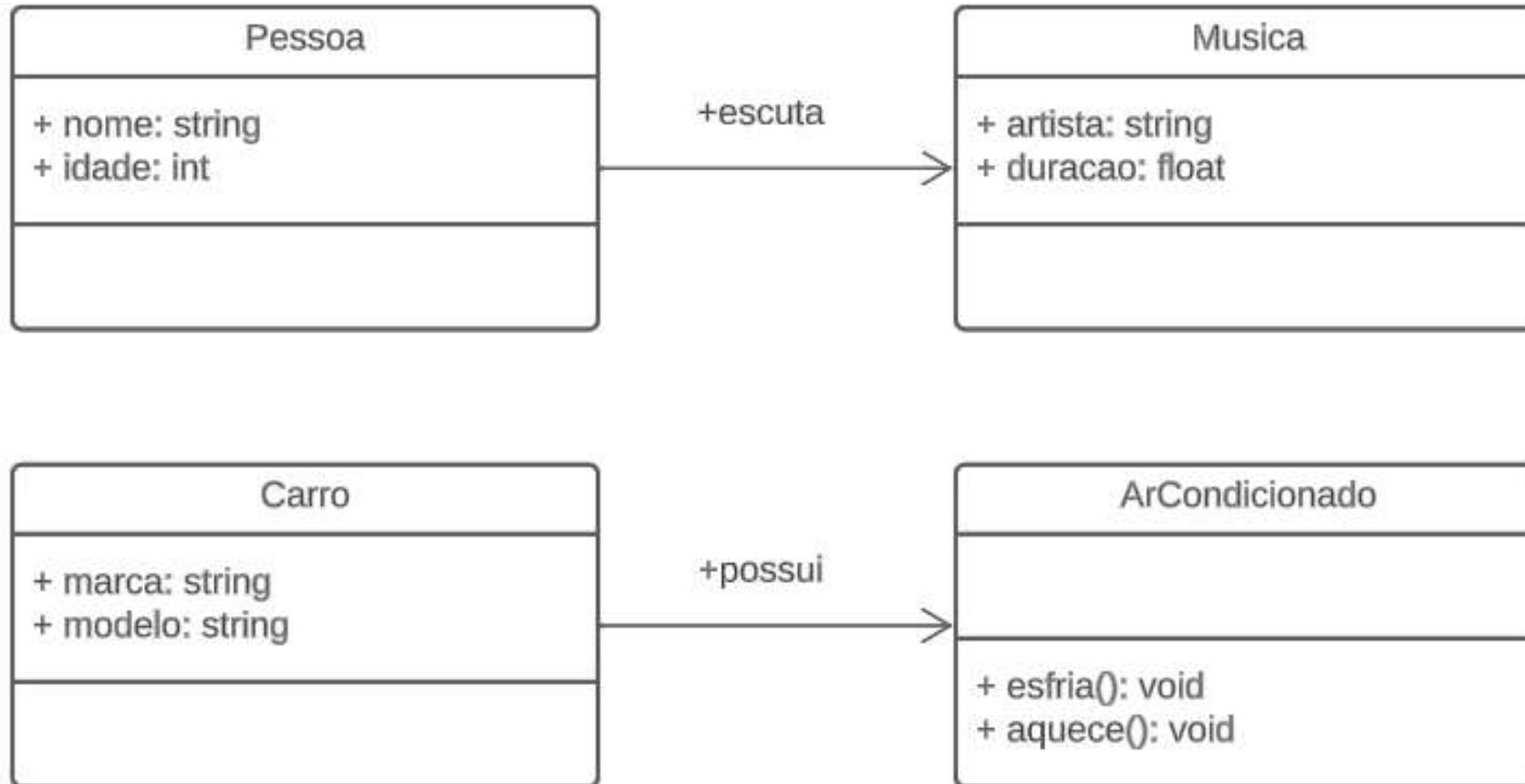
# Relacionamento entre objetos

- ☕ É comum termos classes comunicando com outras classes no sistema.
- ☕ Chamamos essa comunicação de **Associação!!**
- ☕ Uma associação significa **que um objeto contém ou está conectado a outro objeto!** Ocorre nas duas situações abaixo
  - ☕ **Um objeto contiver outro** (o relacionamento “tem um”)
  - ☕ **Um objeto toma ação sobre outro** objeto



# Relacionamento entre objetos

☕ Exemplos:





# Relacionamento entre objetos

☕ As associações ainda podem ser quebradas em:







# Relacionamento entre objetos

- ☕ Imagine expandirmos a nossa classe Conta, **adicionando nome, sobrenome e cpf do cliente.**
- ☕ Sabemos que os “membros da classe” tem um significado.
  - ☕ O que a classe sabe sobre si?
  - ☕ O que uma classe possui?
- ☕ Uma Conta possui um cpf? Ou seria o dono dela que possui o cpf?





# Relacionamento entre objetos

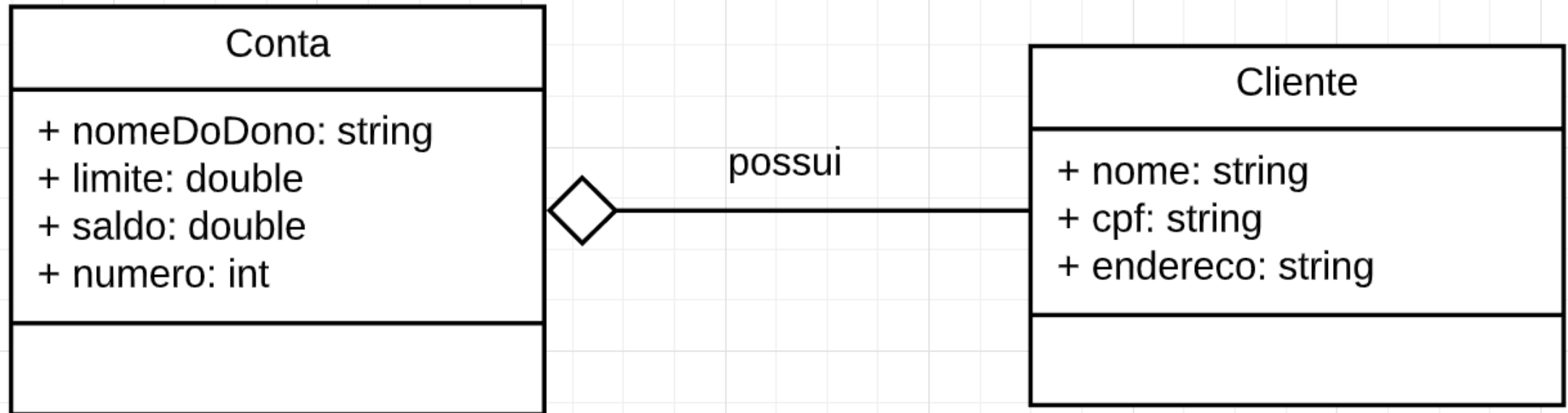
```
public class Conta {  
  
    //Membros da Classe  
    int numero;  
    String nomeDoDono;  
    float saldo;  
    float limite;  
}
```

```
public class Cliente {  
  
    String nome;  
    String endereco;  
    String cpf;  
  
}
```



# Relacionamento entre objetos

☕ Agora que temos uma classe Cliente e uma classe Conta, podemos fazer uma **AGREGAÇÃO** entre elas





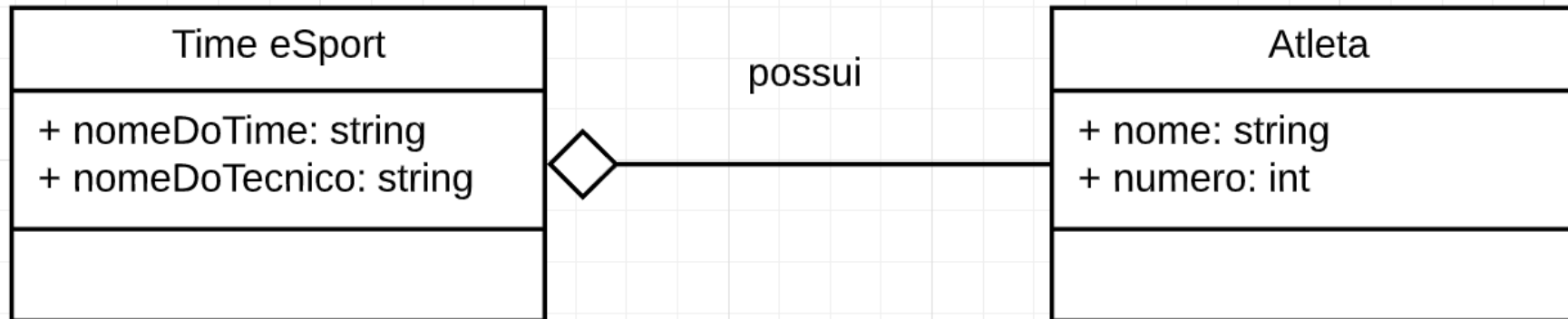
# Relacionamento entre objetos

- ☕ Como podemos definir a **Agregação**?
- ☕ É um tipo de associação **todo/parte**. Na Agregação, a existência do **Objeto-Parte** faz sentido, mesmo não existindo o **Objeto-Todo**.
- ☕ No exemplo Conta-Cliente, o Cliente pode existir independentemente de uma Conta existir.



# Relacionamento entre objetos

## ☕ Outro Exemplo: Time de e-Sport



☕ Os atletas integram o Time eSport, mas podem existir separadamente



# Relacionamento entre objetos

☕ Resgatando a Classe Conta e Cliente

```
public class Conta {  
  
    //Membros da Classe  
    int numero;  
    float saldo;  
    float limite;  
    Cliente titular;
```

```
public class Cliente {  
  
    String nome;  
    String endereco;  
    String cpf;  
  
}
```



# Relacionamento entre objetos

## Resgatando a Classe Conta e Cliente

```
public class Main {  
  
    public static void main(String[] args) {  
        Conta conta = new Conta();  
        Cliente cliente = new Cliente();  
  
        conta.titular = cliente;  
  
    }  
  
}
```



# Relacionamento entre objetos

- ☕ Agora a variável “cliente” e “conta.titular” guardam uma referência para o mesmo lugar na memória.
- ☕ Com ambas essas variáveis podemos acessar esse “cliente”
- ☕ E podemos usar o operador “.” para navegar sobre a estrutura de classes





```
Conta conta = new Conta();  
Cliente cliente = new Cliente();
```

```
cliente.nome = "Jaum";  
cliente.cpf = "123.123.123";
```

```
conta.titular = cliente;
```

```
System.out.println("Nome do cliente: " + conta.titular.nome);  
System.out.println("CPF do cliente: " + conta.titular.cpf);
```

```
//Mesma saída
```

```
System.out.println("Nome do cliente: " + cliente.nome);
```



# Relacionamento entre objetos

☕ E se tentássemos acessar o titular (Cliente) sem fazer uma atribuição antes? Ou a criação?

```
Conta conta = new Conta();  
Cliente cliente = new Cliente();
```

```
cliente.nome = "Jaum";  
cliente.cpf = "123.123.123";
```

```
//conta.titular = cliente;
```

```
System.out.println("Nome do cliente: " + conta.titular.nome);  
System.out.println("CPF do cliente: " + conta.titular.cpf);
```



# Relacionamento entre objetos

☕ Teríamos um “NullPointerException”

☕ Pois o objeto não foi inicializado!

```
Exception in thread "main" java.lang.NullPointerException  
    at Main.main(Main.java:12)
```



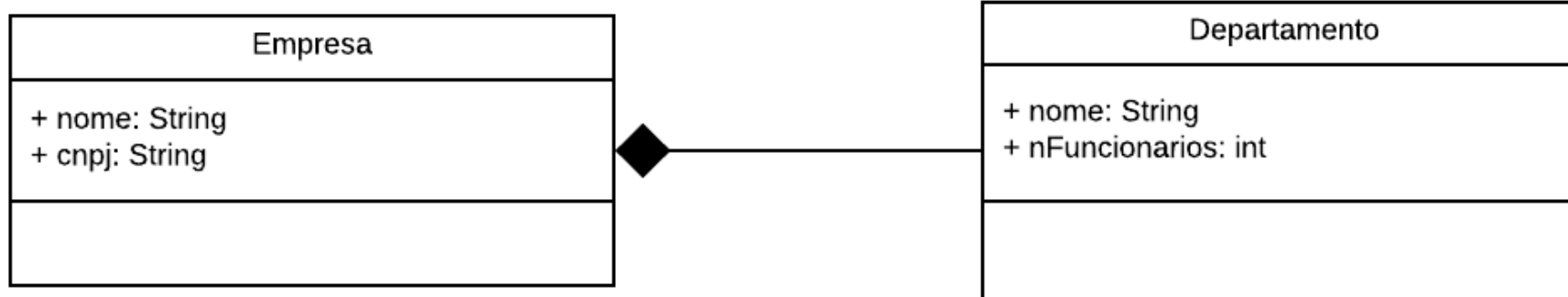
# Relacionamento entre objetos

- ☕ Nos exemplos anteriores, utilizamos a Agregação!
- ☕ Outro conceito é o de **COMPOSIÇÃO**
- ☕ Também é uma associação **Todo/Parte**, e funciona como uma agregação mais forte. **Nela, a existência do Objeto-Parte não faz sentido se o Objeto-Todo não existir**



# Relacionamento entre objetos

☕ Exemplo: Uma empresa com seus Departamentos





# Relacionamento entre objetos

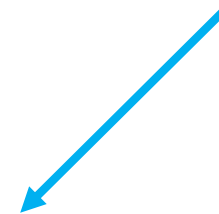
- ☕ Na composição, o Objeto-Todo é responsável pelas suas partes.
- ☕ Ele é responsável pela criação e destruição

```
public class Empresa {
```

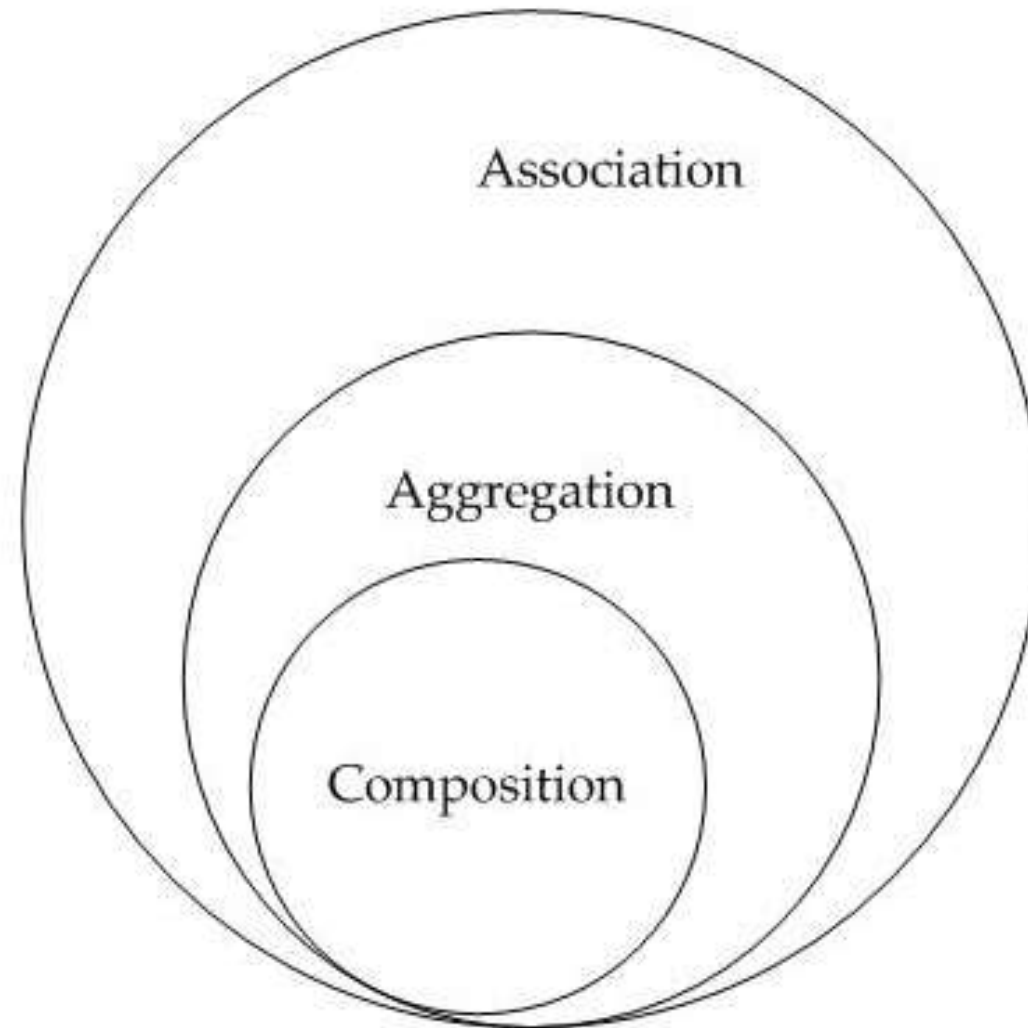
```
    String nome;  
    Departamento dep;
```

```
    public Empresa() {  
        dep = new Departamento();  
    }
```

Construtor



# Relacionamento entre objetos





# Lista de exercícios

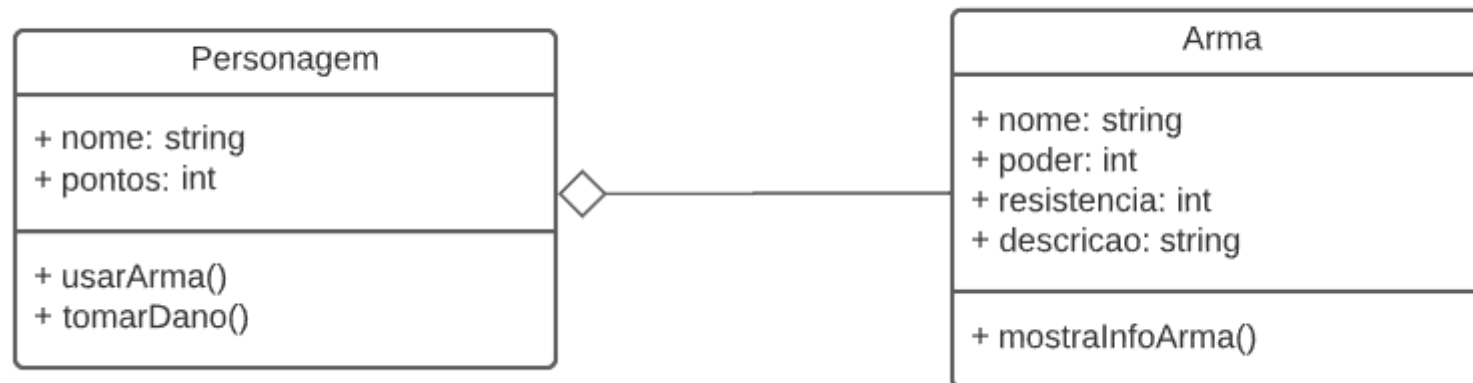




## Exercício 5

### Exercício 5 – Jogo!

- Crie Classes em Java que atendam a seguinte especificação
- Cada vez que o personagem toma dano ele perde 5 pontos de vida;
- Cada vez que usa a arma, ela perde 2 pontos de resistência

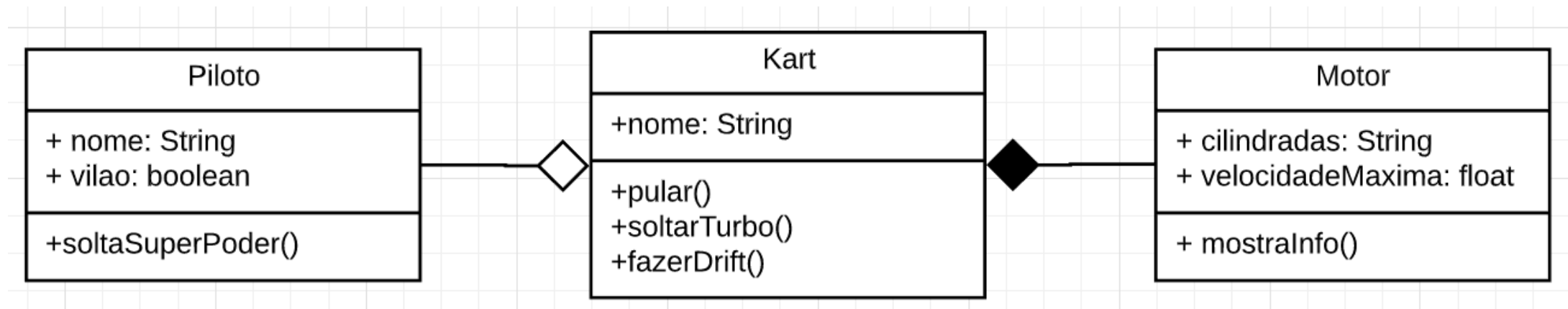




# Exercício 6

## ☕ Exercício 6 – Mario Kart!

- ☕ Crie Classes em Java que atendam a especificação UML abaixo
- ☕ Em seguida crie ao menos dois objetos de cada, os relacione e
- ☕ Os métodos devem apenas imprimir suas ações
- ☕ As cilindradas podem assumir três valores: 50, 100 e 150
- ☕ A velocidade pode assumir valores entre 0 e 150



# Resolução dos Exercícios



[https://github.com/chrislima-inatel/C206\\_C125](https://github.com/chrislima-inatel/C206_C125)





# Material Complementar

- ☕ Capítulo 4 da apostila FJ-11
  - ☕ Orientação a Objetos Básica