

*introdução à*  
**ORIENTAÇÃO**  
*a* **OBJETOS #4**

BY RAFA

# *introdução à* **ORIENTAÇÃO** *a* **OBJETOS** #4

**REVISÃO**

**HERANÇA**

**POLIMORFISMO**

# *programação* **ORIENTADA** *a* **OBJETOS**



A diagram showing four OOP concepts arranged in a 2x2 grid. The top row contains 'ABSTRAÇÃO' and 'ENCAPSULAMENTO', and the bottom row contains 'HERANÇA' and 'POLIMORFISMO'. Each concept is in a light gray box with a black border. Above each box is a small colored square: pink for 'ABSTRAÇÃO', teal for 'ENCAPSULAMENTO', yellow for 'HERANÇA', and pink for 'POLIMORFISMO'. The word 'HERANÇA' is circled in pink.

**ABSTRAÇÃO**

**ENCAPSULAMENTO**

**HERANÇA**

**POLIMORFISMO**

# HERANÇA

# OBJETO

Sandijunior
raça: SRD cor: preto
late(): void corre(): void fazCoco(): Coco



Fred
raça: Beagle cor: bege
late(): void corre(): void fazCoco(): Coco

# HERANÇA

# OBJETO

Sandijunior
raça: SRD cor: preto
late(): void corre(): void fazCoco(): Coco

Fred
raça: Beagle cor: bege
late(): void corre(): void fazCoco(): Coco

Cachorro
raça: string cor: string
late(): void corre(): void fazCoco(): Coco

# HERANÇA

## OBJETO

Sandijunior
raça: SRD cor: preto
late(): void corre(): void fazCoco(): Coco

Fred
raça: Beagle cor: bege
late(): void corre(): void fazCoco(): Coco

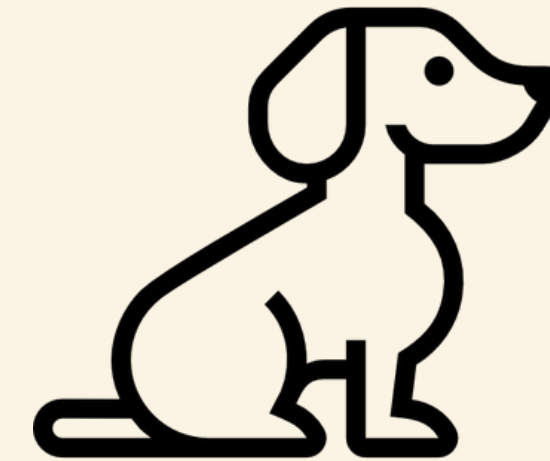
## CLASSE

Cachorro
raça: string cor: string
late(): void corre(): void fazCoco(): Coco

# HERANÇA

## CLASSE

Cachorro
raça: string cor: string
late(): void corre(): void fazCoco(): Coco



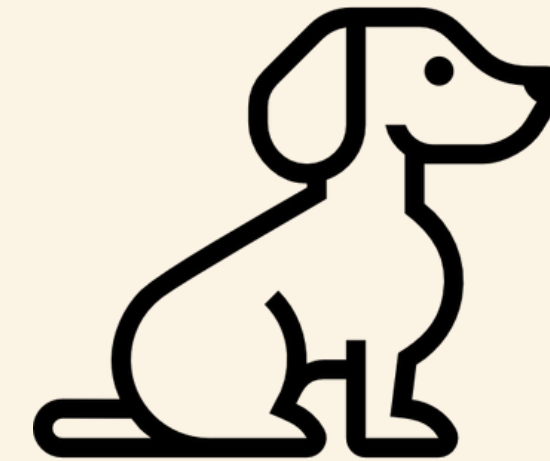
Classificação científica	
Reino:	Animalia
Filo:	Chordata
Classe:	Mammalia
Ordem:	Carnivora
Subordem:	Caniformia
Família:	Canidae
Gênero:	<i>Canis</i>
Espécie:	<i>C. lupus</i>
Subespécie:	<i>C. l. familiaris</i>

BY WIKIPEDIA

# HERANÇA

## CLASSE

Cachorro
raça: string cor: string
late(): void corre(): void fazCoco(): Coco



### Classificação científica

Reino:	Animalia
Filo:	Chordata
Classe:	Mammalia
Ordem:	Carnivora
Subordem:	Caniformia
Família:	Canidae
Gênero:	<i>Canis</i>
Espécie:	<i>C. lupus</i>
Subespécie:	<i>C. l. familiaris</i>

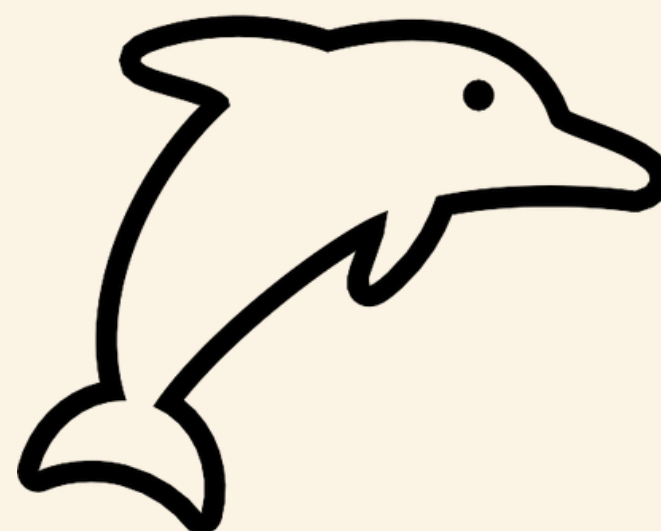
BY WIKIPEDIA



# HERANÇA



Classificação científica	
Reino:	Animalia
Filo:	Chordata
Classe:	Mammalia
Ordem:	Carnivora
Subordem:	Caniformia
Família:	Canidae
Gênero:	<i>Canis</i>
Espécie:	<i>C. lupus</i>
Subespécie:	<i>C. l. familiaris</i>



Classificação científica	
Reino:	Animalia
Filo:	Chordata
Classe:	Mammalia
Ordem:	Cetacea
Subordem:	Odontoceti
Família:	<b>Delphinidae</b> Gray, 1821



Classificação científica	
Reino:	Animalia
Filo:	Chordata
Classe:	Mammalia
Infraclasse:	Placentalia
Ordem:	<b>Chiroptera</b> Blumenbach, 1779

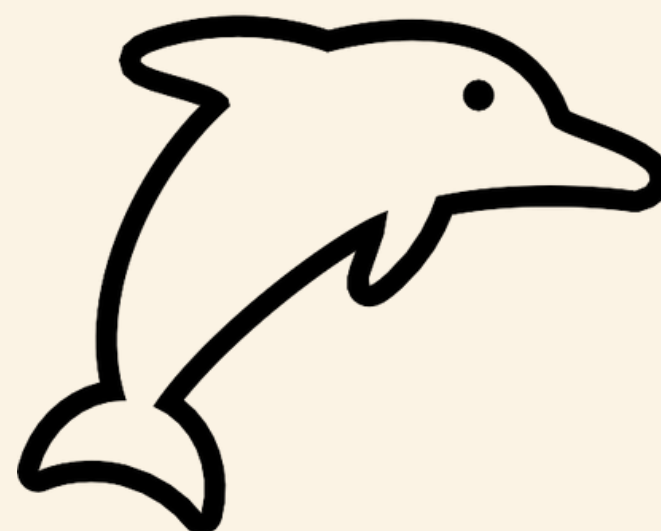
BY WIKIPEDIA

# HERANÇA



## Classificação científica

Reino: Animalia  
Filo: Chordata  
Classe: Mammalia  
Ordem: Carnivora  
Subordem: Caniformia  
Família: Canidae  
Gênero: *Canis*  
Espécie: *C. lupus*  
Subespécie: *C. l. familiaris*



## Classificação científica

Reino: Animalia  
Filo: Chordata  
Classe: Mammalia  
Ordem: Cetacea  
Subordem: Odontoceti  
Família: **Delphinidae**  
Gray, 1821



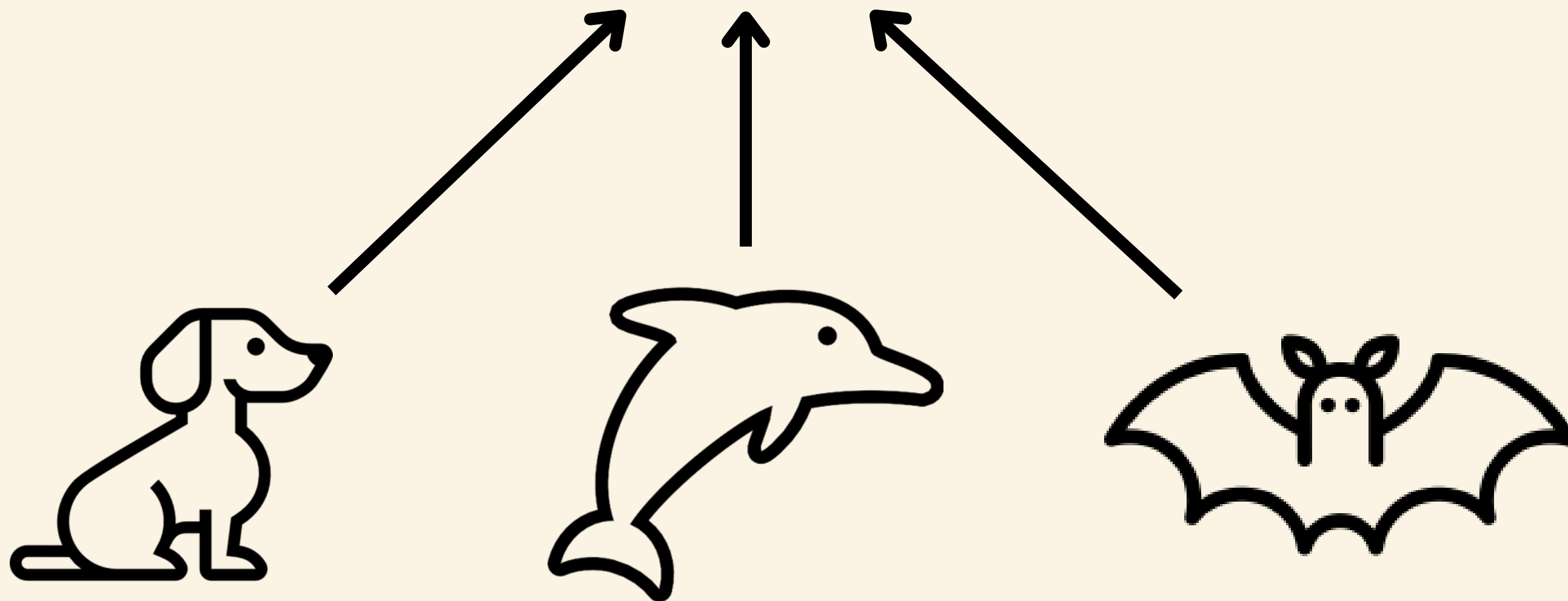
## Classificação científica

Reino: Animalia  
Filo: Chordata  
Classe: Mammalia  
Infraclasse: Placentalia  
Ordem: **Chiroptera**  
Blumenbach, 1779

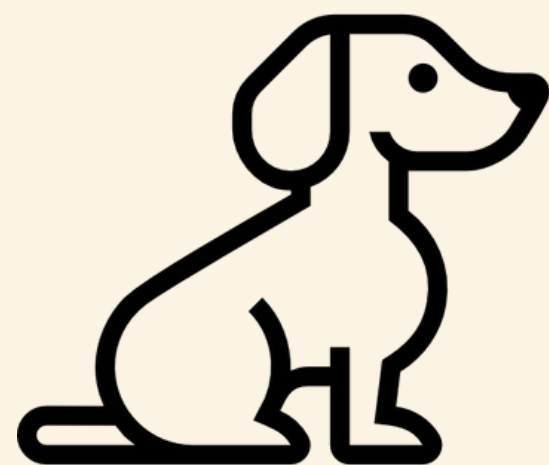
BY WIKIPEDIA

HERANÇA

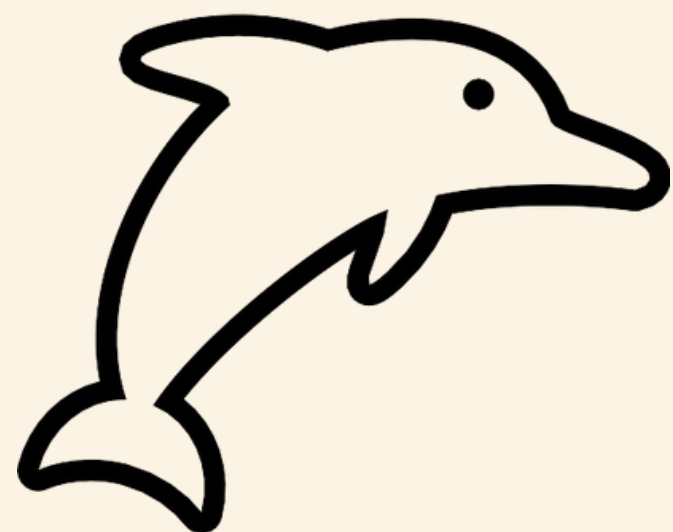
# MAMÍFERO



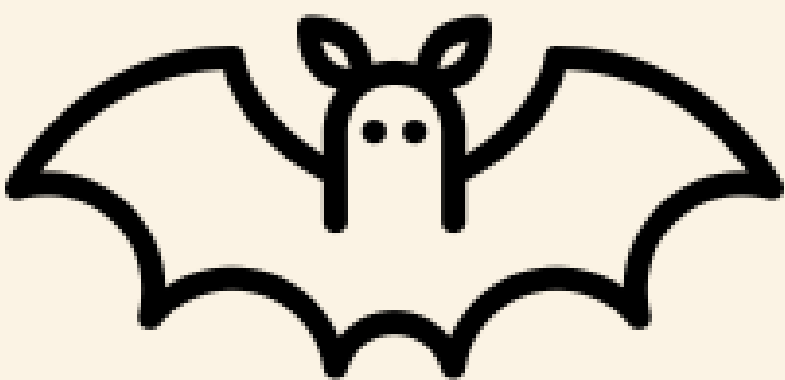
# HERANÇA



Cachorro
raça: string cor: string
late(): void corre(): void amamenta(): void fazCoco(): Coco

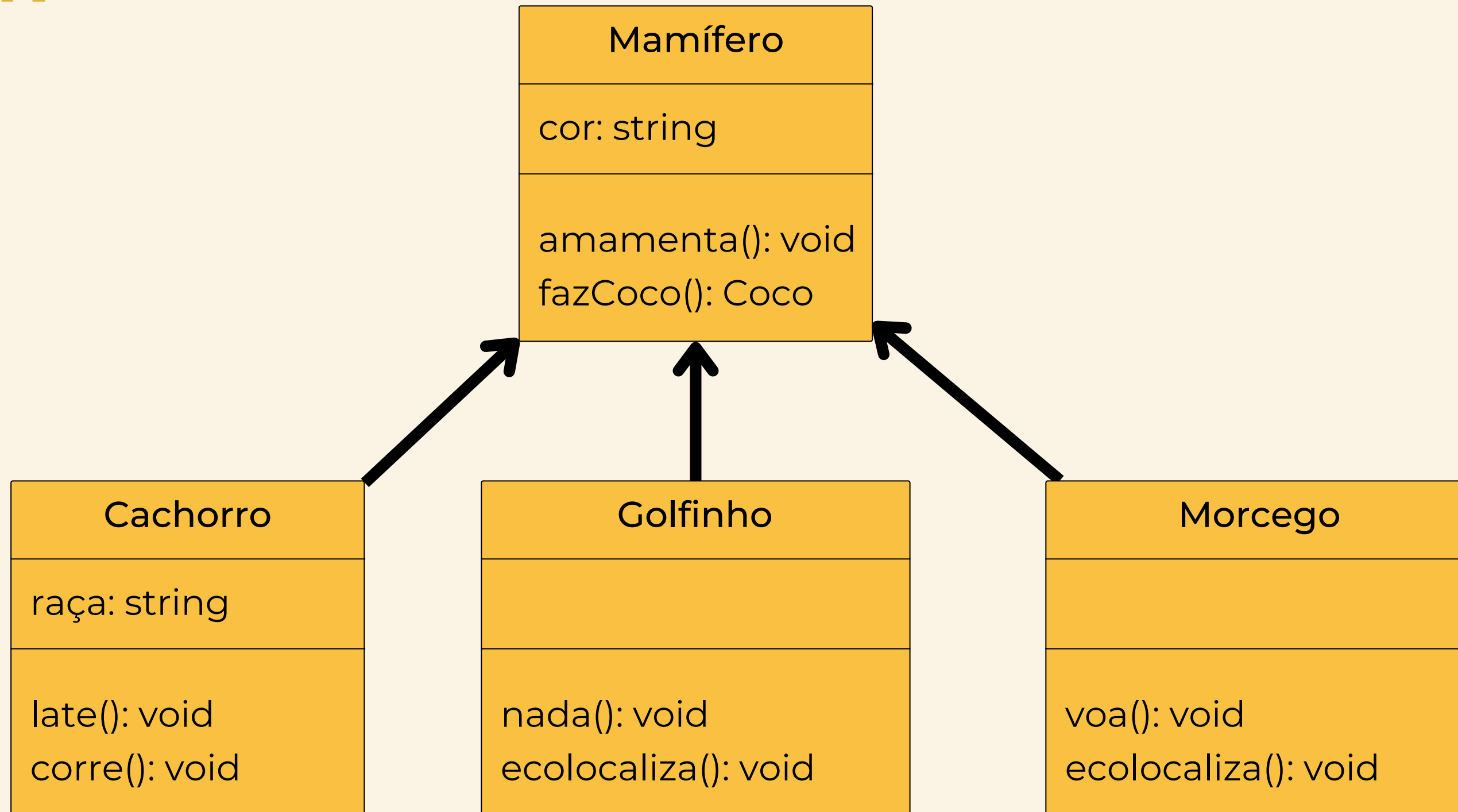


Golfinho
cor: string
nada():void ecolocaliza(): void amamenta(): void fazCoco(): Coco

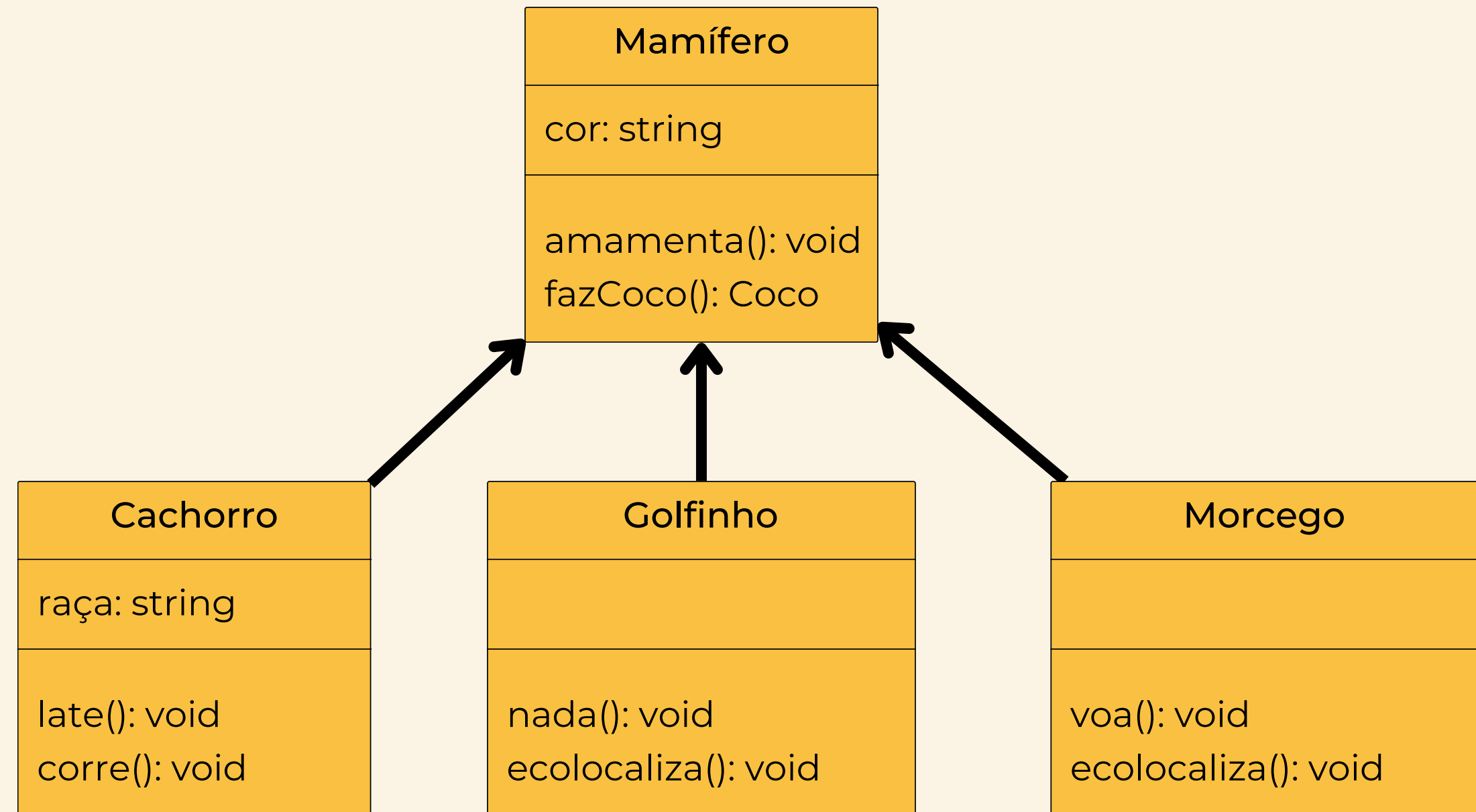


Morcego
cor: string
voa():void ecolocaliza(): void amamenta(): void fazCoco(): Coco

# HERANÇA



# HERANÇA



Uma classe, ao ter classes que a herdam é dita como classe **PAI/MÃE** ou **SUPER CLASSE** enquanto as classes que a herdam são ditas como classes **FILHAS**



Permite o **APROVEITAMENTO** de propriedades de outras classes

# HERANÇA

```
class Mamifero {  
    cor: string;  
  
    constructor(cor: string) {  
        this.cor = cor;  
    }  
  
    amamenta(): void {  
        // implementação do método  
    }  
  
    fazCoco(): Coco {  
        // implementação do método  
    }  
}
```

```
class Cachorro extends Mamifero {  
    raca: string;  
  
    constructor(cor: string, raca: string) {  
        super(cor)  
        this.raca = raca;  
    }  
  
    late(): void {  
        // implementação do método  
    }  
  
    corre(): void {  
        // implementação do método  
    }  
}
```

# HERANÇA

classe mãe

```
class Mamifero {  
  cor: string;  
  
  constructor(cor: string) {  
    this.cor = cor;  
  }  
  
  amamenta(): void {  
    // implementação do método  
  }  
  
  fazCoco(): Coco {  
    // implementação do método  
  }  
}
```

```
class Cachorro extends Mamifero {  
  raca: string;  
  
  constructor(cor: string, raca: string) {  
    super(cor)  
    this.raca = raca;  
  }  
  
  late(): void {  
    // implementação do método  
  }  
  
  corre(): void {  
    // implementação do método  
  }  
}
```



# HERANÇA

classe mãe

```
class Mamifero {  
  cor: string;  
  
  constructor(cor: string) {  
    this.cor = cor;  
  }  
  
  amamenta(): void {  
    // implementação do método  
  }  
  
  fazCoco(): Coco {  
    // implementação do método  
  }  
}
```

classe filha

```
class Cachorro extends Mamifero {  
  raca: string;  
  
  constructor(cor: string, raca: string) {  
    super(cor)  
    this.raca = raca;  
  }  
  
  late(): void {  
    // implementação do método  
  }  
  
  corre(): void {  
    // implementação do método  
  }  
}
```

# HERANÇA

classe mãe

```
class Mamifero {  
  cor: string;  
  
  constructor(cor: string) {  
    this.cor = cor;  
  }  
  
  amamenta(): void {  
    // implementação do método  
  }  
  
  fazCoco(): Coco {  
    // implementação do método  
  }  
}
```

chamada do  
construtor da  
classe mãe

classe filha

```
class Cachorro extends Mamifero {  
  raca: string;  
  
  constructor(cor: string, raca: string) {  
    super(cor)  
    this.raca = raca;  
  }  
  
  late(): void {  
    // implementação do método  
  }  
  
  corre(): void {  
    // implementação do método  
  }  
}
```

# *programação* **ORIENTADA** *a* **OBJETOS**



**ABSTRAÇÃO**



**ENCAPSULAMENTO**



**HERANÇA**



**POLIMORFISMO**

# POLIMORFISMO

Mamífero
cor: string
amamenta(): void fazCoco(): Coco

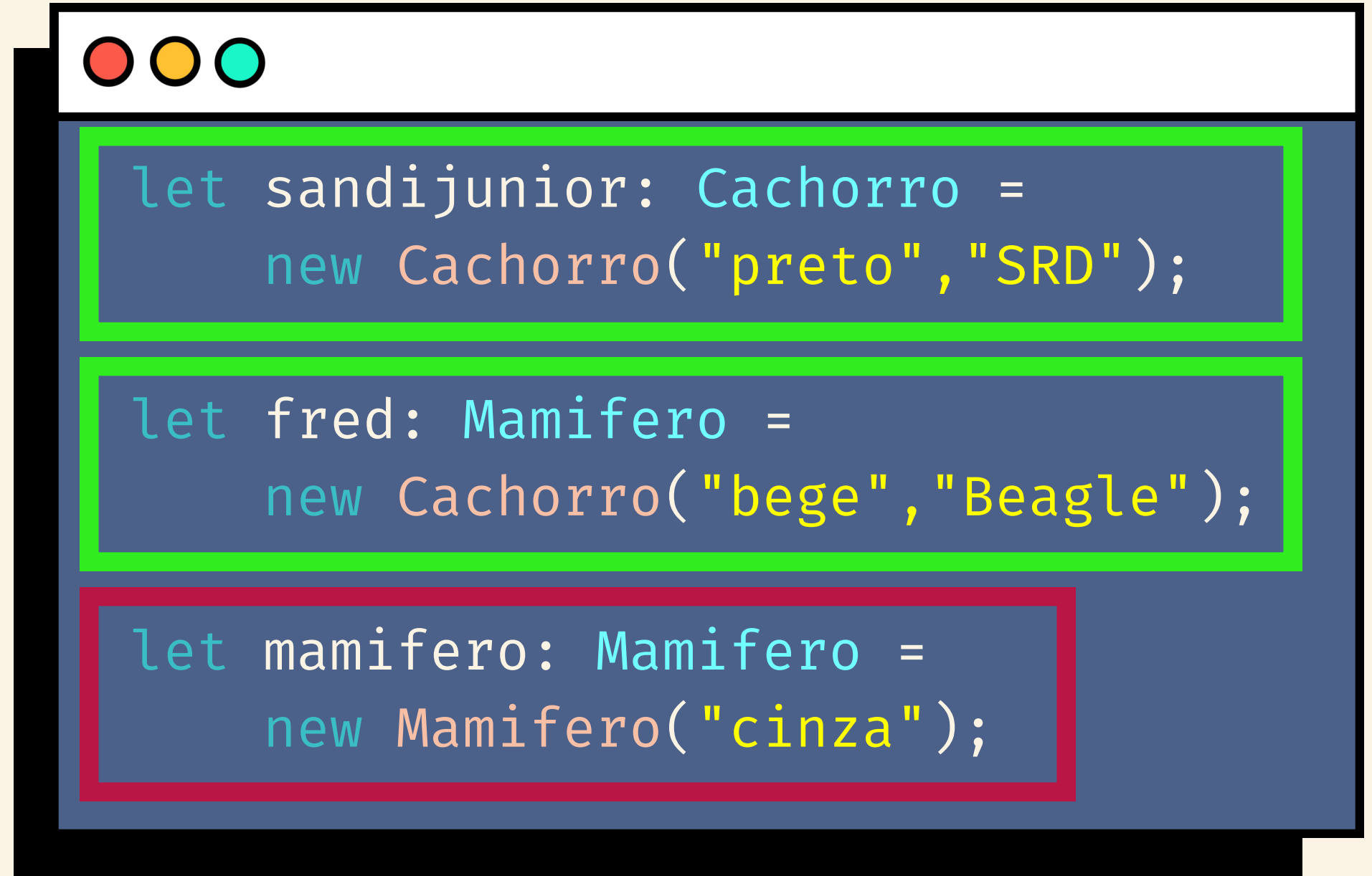
# POLIMORFISMO

Mamífero
cor: string
amamenta(): void fazCoco(): Coco

```
let sandijunior: Cachorro =  
    new Cachorro("preto", "SRD");  
  
let fred: Mamifero =  
    new Cachorro("bege", "Beagle");  
  
let mamifero: Mamifero =  
    new Mamifero("cinza");
```

# POLIMORFISMO

Mamífero
cor: string
amamenta(): void fazCoco(): Coco



```
let sandijunior: Cachorro =  
    new Cachorro("preto", "SRD");  
  
let fred: Mamifero =  
    new Cachorro("bege", "Beagle");  
  
let mamifero: Mamifero =  
    new Mamifero("cinza");
```

Não PODE ocorrer uma  
instância de Mamífero

# POLIMORFISMO

<div>&lt;abstract&gt;</div> <div>Mamífero</div>
<div>cor: string</div>
<div>amamenta(): void</div> <div>fazCoco(): Coco</div>

# POLIMORFISMO

<abstract> <b>Mamífero</b>
cor: string
amamenta(): void fazCoco(): Coco



São classes que representam conceitos tão genéricos que não vale a pena trabalhar com eles **DIRETAMENTE**.



Não permite o uso do construtor **SOZINHA**.  
**PRECISA** de uma extensão/classe filha para ser utilizada.



# POLIMORFISMO

```
abstract class Mamifero {  
    cor: string;  
  
    constructor(cor: string) {  
        this.cor = cor;  
    }  
  
    amamenta(): void {  
        // implementação do método  
    }  
  
    fazCoco(): Coco {  
        // implementação do método  
    }  
}
```

```
class Cachorro extends Mamifero {  
    raca: string;  
  
    constructor(cor: string, raca: string) {  
        super(cor)  
        this.raca = raca;  
    }  
  
    late(): void {  
        // implementação do método  
    }  
  
    corre(): void {  
        // implementação do método  
    }  
}
```

# POLIMORFISMO

Mamífero
cor: string
amamenta(): void fazCoco(): Coco desloca(): void comunica(): void

# POLIMORFISMO

```
abstract class Mamifero {
    cor: string;

    constructor(cor: string) {
        this.cor = cor;
    }

    amamenta(): void {
        // implementação do método
    }

    fazCoco(): Coco {
        // implementação do método
    }

    abstract comunica(): void;
    abstract desloca(): void;
}
```

```
class Cachorro extends Mamifero {
    raca: string;

    constructor(cor: string, raca: string) {
        super(cor)
        this.raca = raca;
    }

    private late(): void {
        // implementação do método
    }

    private corre(): void {
        // implementação do método
    }

    comunica(): void {
        this.late();
    }

    desloca(): void {
        this.corre();
    }
}
```

# POLIMORFISMO

```
abstract class Mamifero {
    cor: string;

    constructor(cor: string) {
        this.cor = cor;
    }

    amamenta(): void {
        // implementação do método
    }

    fazCoco(): Coco {
        // implementação do método
    }

    abstract comunica(): void;
    abstract desloca(): void;
}
```

```
class Cachorro extends Mamifero {
    raca: string;

    constructor(cor: string, raca: string) {
        super(cor)
        this.raca = raca;
    }

    private late(): void {
        // implementação do método
    }

    private corre(): void {
        // implementação do método
    }

    comunica(): void {
        this.late();
    }

    desloca(): void {
        this.corre();
    }
}
```

```
class Morcego extends Mamifero {
    constructor(cor: string) {
        super(cor)
    }

    private ecolocaliza(): void {
        // implementação do método
    }

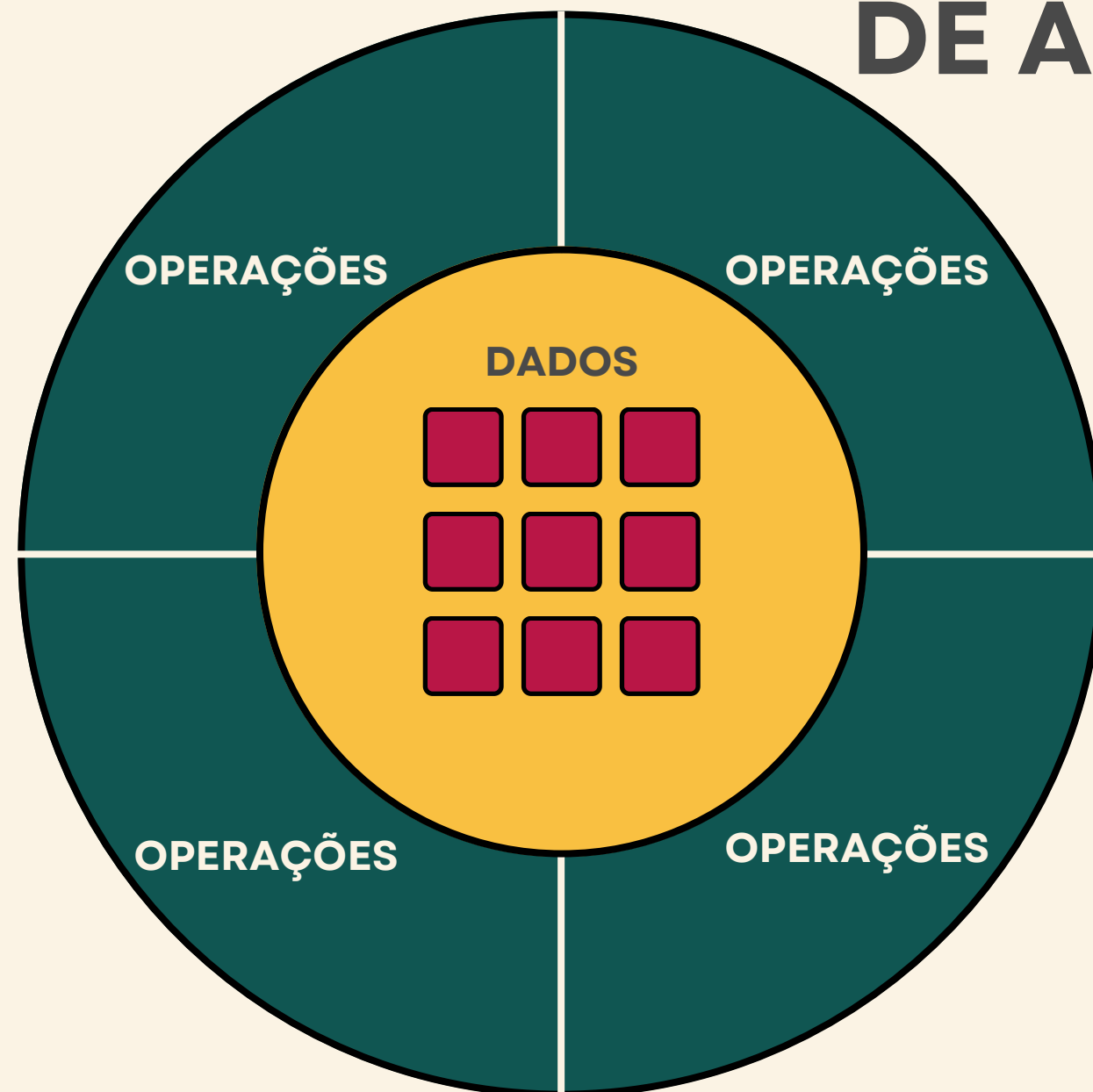
    private voa(): void {
        // implementação do método
    }

    comunica(): void {
        this.ecolocaliza();
    }

    desloca(): void {
        this.voa();
    }
}
```

# ENCAPSULAMENTO

## MODIFICADORES DE ACESSO



### **PUBLIC**

Modificador de acesso PADRÃO. Permite o acesso INDISCRIMINADO.



### **PRIVATE**

Define o acesso PRIVADO a VARIÁVEIS e MÉTODOS permitindo o acesso apenas INTERNO pela própria classe.

# ENCAPSULAMENTO

## MODIFICADORES DE ACESSO



# ENCAPSULAMENTO

## MODIFICADORES DE ACESSO



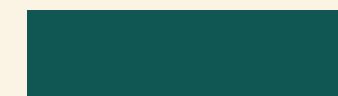
### PUBLIC

Modificador de acesso PADRÃO. Permite o acesso INDISCRIMINADO.



### PRIVATE

Define o acesso PRIVADO a VARIÁVEIS e MÉTODOS permitindo o acesso apenas INTERNO pela própria classe.










### PROTECTED

Utilizado APENAS no contexto de HERANÇA. Permite o acesso interno apenas à própria CLASSE e suas FILHAS.

# ENCAPSULAMENTO

## MODIFICADORES DE ACESSO

MODIFICADOR	ACESSO INTERNO	ACESSO FILHAS	ACESSO EXTERNO
PRIVATE			
PROTECTED			
PUBLIC			



# DESAFIO

## COMBATE

## RPG

**OBRIGADO!**

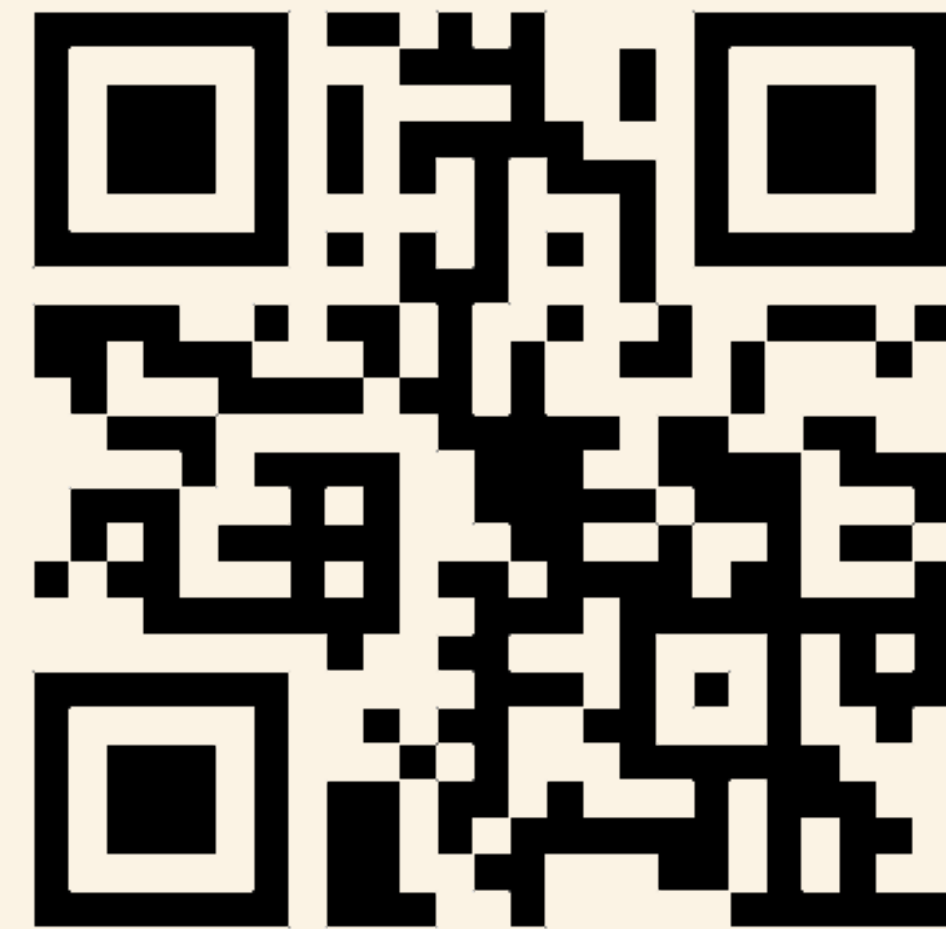
# FEEDBACK

**ACESSE**

**WWW.MENTI.COM**

**INSIRA O CÓDIGO**

**6363 6390**



ou use o QR code