

introdução à
ORIENTAÇÃO
a **OBJETOS #5**

BY RAFA

introdução à **ORIENTAÇÃO** *a* **OBJETOS** **#5**

REVISÃO

INFORMAÇÃO EXTRA

POLIMORFISMO #2

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



ABSTRAÇÃO



ENCAPSULAMENTO



HERANÇA



POLIMORFISMO

ABSTRAÇÃO

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



OBJETO

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

ABSTRAÇÃO

OBJETO

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

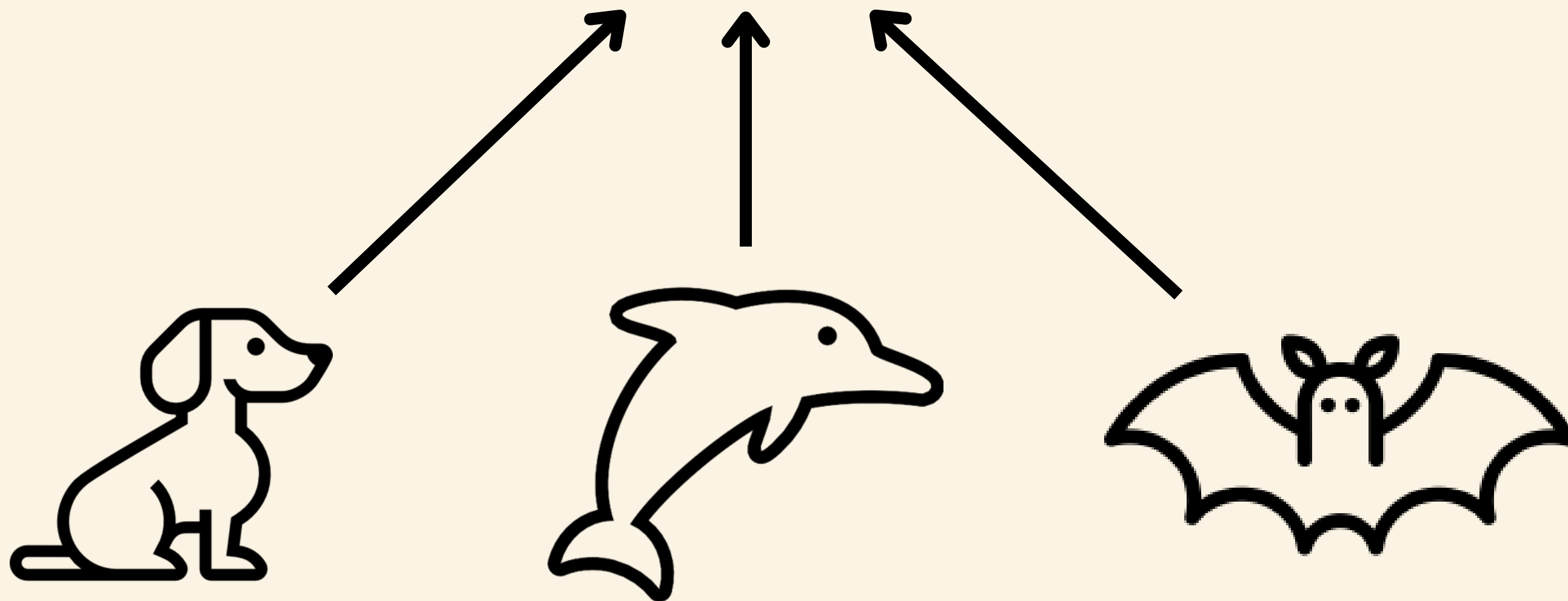
Fred
raça: Beagle cor: bege
late(): void corre(): void fazCoco(): Coco

CLASSE

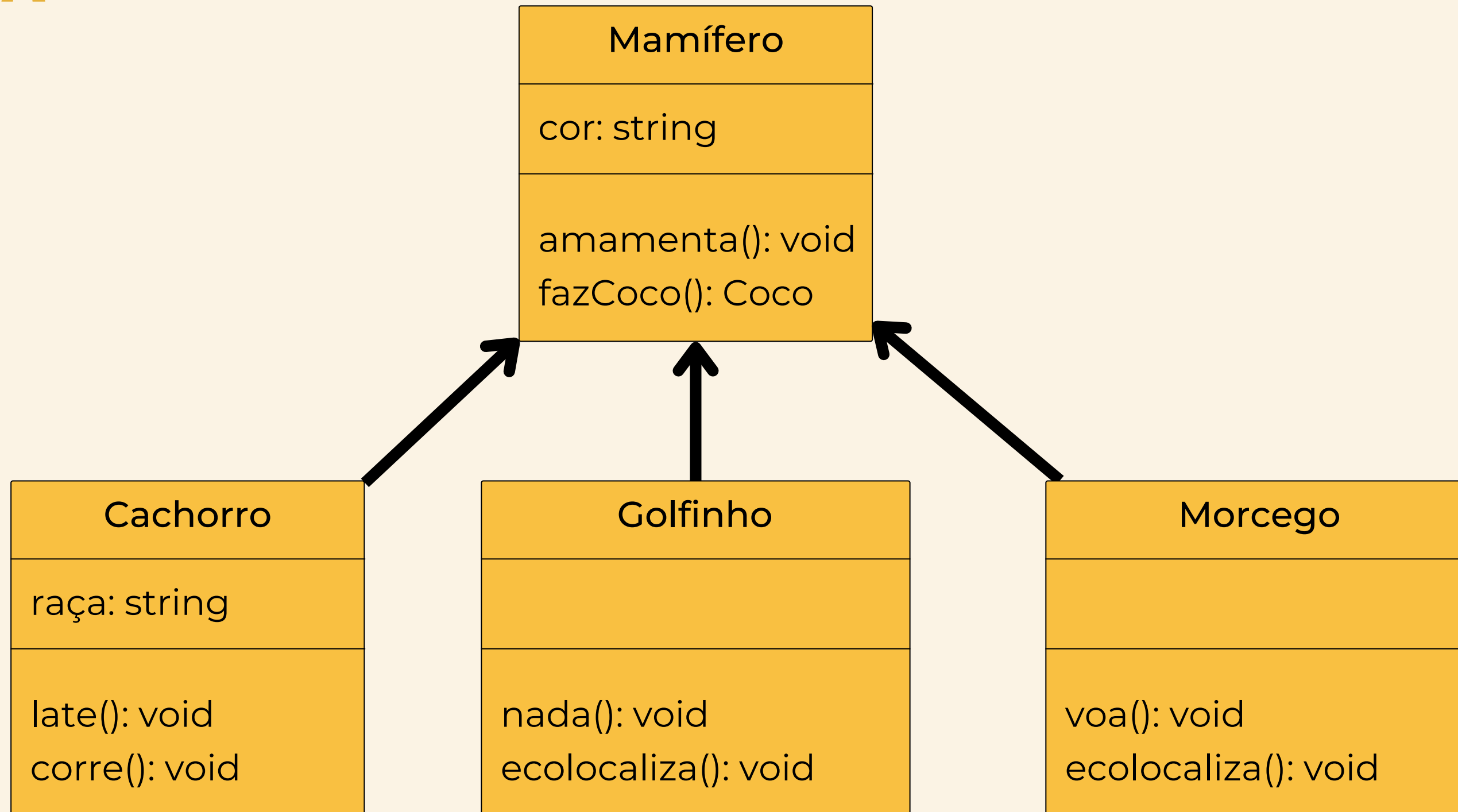
Cachorro
raça: string cor: string
late(): void corre(): void fazCoco(): Coco

HERANÇA

MAMÍFERO



HERANÇA



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



ENCAPSULAMENTO

MODIFICADORES DE ACESSO

MODIFICADOR	ACESSO INTERNO	ACESSO FILHAS	ACESSO EXTERNO
PRIVATE			
PROTECTED			
PUBLIC			

INFORMAÇÃO

EXTRA

INFORMAÇÃO EXTRA



THIS

Uma referência à **instância** corrente, ou seja, ao **objeto** em questão. Usado para referenciar os **atributos** e **métodos** daquela instância.

```
class Thing {
  protected name: string;
  protected life: number;
  protected destroyed: boolean;

  constructor(name: string,
               life: number) {
    this.life = life;
    this.destroyed = false;
    this.name = name;
  }

  destroy() {
    this.life = 0;
    this.destroyed = true;
  }
}
```

GET/SET

Utilizado para **expor atributos protegidos** por modificadores de acesso. É importante que seja **usado com cautela**. Não há sentido em um modificador de acesso se implementados ambos get/set públicos.

```
class Thing {  
    protected name: string;  
    protected life: number;  
    protected destroyed: boolean;  
  
    constructor(name: string,  
                life: number) {  
        this.life = life;  
        this.destroyed = false;  
        this.name = name;  
    }  
  
    destroy(): void {  
        this.life = 0;  
        this.destroyed = true;  
    }  
  
    getLife(): number {  
        return this.life;  
    }  
  
    takeDamage(damage: number): void {  
        this.life -= damage;  
        if (this.life <= 0) {  
            this.destroy();  
        }  
    }  
}
```

GET/SET

Utilizado para **expor atributos protegidos** por modificadores de acesso. É importante que seja **usado com cautela**. Não há sentido em um modificador de acesso se implementados ambos get/set públicos.

ATENÇÃO: getters para **atributos** do tipo **boolean** possuem o **prefixo "is"** ao invés de "get".

```
class Thing {  
    protected name: string;  
    protected life: number;  
    protected destroyed: boolean;  
  
    constructor(name: string,  
                life: number) {  
        this.life = life;  
        this.destroyed = false;  
        this.name = name;  
    }  
  
    destroy(): void {  
        this.life = 0;  
        this.destroyed = true;  
    }  
  
    getLife(): number {  
        return this.life;  
    }  
  
    takeDamage(damage: number): void {  
        this.life -= damage;  
        if (this.life <= 0) {  
            this.destroy();  
        }  
    }  
  
    isDestroyed(): boolean {  
        return this.destroyed;  
    }  
}
```

EQUALS

Método utilizado para **comparar** dois **objetos**.

Ao utilizar objetos, deve-se **evitar** a comparação por meio dos operadores "==" e "===". Apesar de funcionar, estes operadores **não comparam o conteúdo** dos objetos e sim o **enderço de memória** ocupado por ele.

Implementando um método **equals** é possível seguir a **regra** cabível ao contexto da aplicação.

```
class Thing {  
    protected name: string;  
    protected life: number;  
    protected destroyed: boolean;  
  
    constructor(name: string,  
                life: number) {  
        this.life = life;  
        this.destroyed = false;  
        this.name = name;  
    }  
  
    //métodos  
  
    getName(): string {  
        return this.name;  
    }  
  
    equals(thing: Thing): boolean {  
        return this.name === thing.getName()  
    }  
}
```


CONSTANTES

Tal qual o tipo **const** do **javascript**, estes valores **não** podem ser **alterados** em tempo de **execução**.

readonly utilizado para definir um **atributo** cujo valor pode ser alterado **apenas** no **construtor**.

static operador utilizado para definir um **atributo** que **não** será **acessível** pelas **instâncias**. Será **visível** apenas a nível da **classe**.

```
class Thing {
  static readonly defaultLife = 1000;

  protected name: string;
  protected life: number;
  protected destroyed: boolean;

  constructor(name: string,
               life: number = Thing.defaultLife) {
    this.life = life;
    this.destroyed = false;
    this.name = name;
  }

  //métodos
}
```

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



ABSTRAÇÃO



ENCAPSULAMENTO



HERANÇA



POLIMORFISMO

POLIMORFISMO

OBJETO

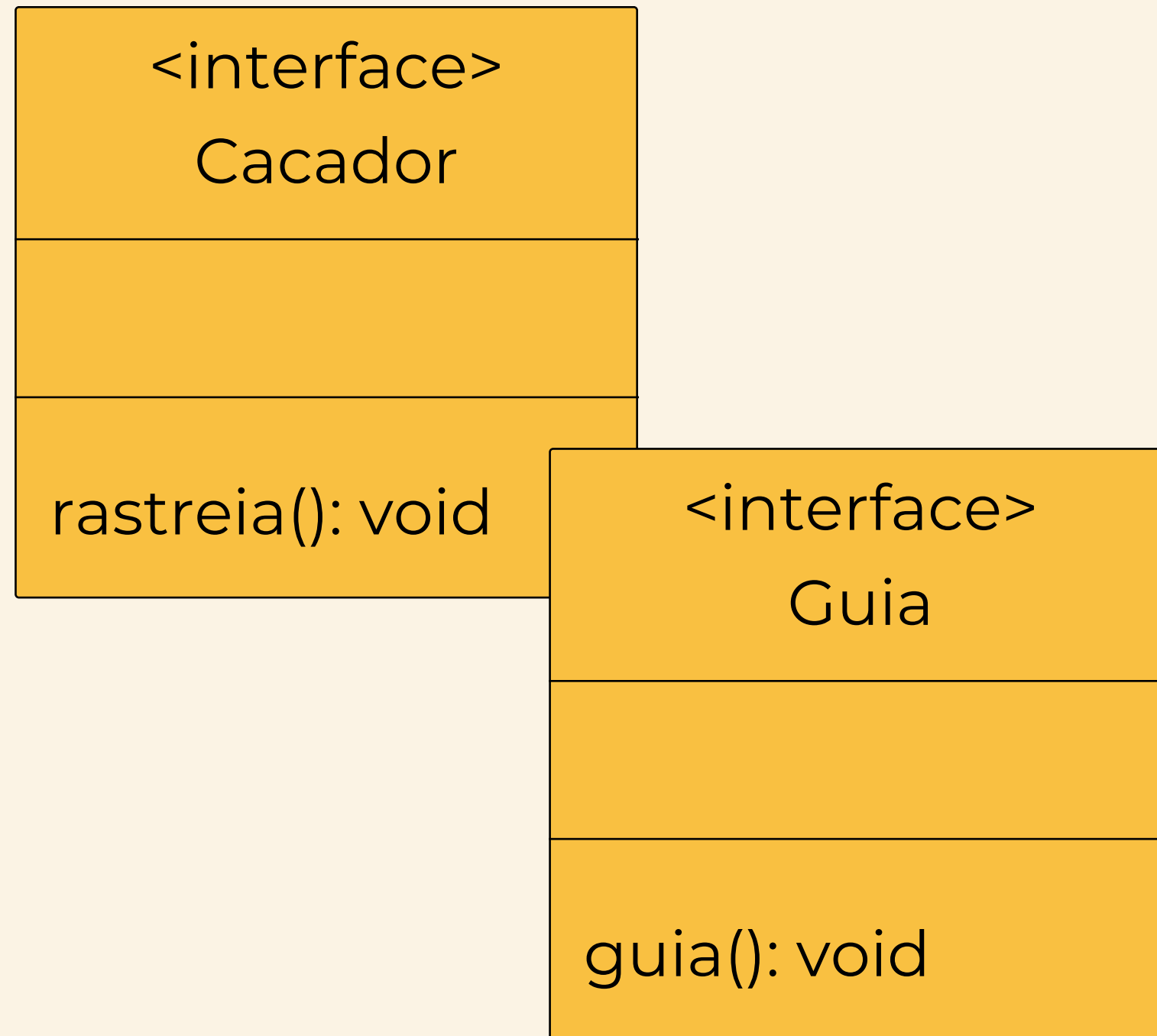
Sandijunior
raça: SRD cor: preto
late(): void corre(): void fazCoco(): Coco

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

CLASSE

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

POLIMORFISMO



INTERFACES podem ser vistas como classes **ABSTRATAS** mais **LIMITADAS**. Também não podem ser instanciadas diretamente.



NÃO permite a **IMPLEMENTAÇÃO** de métodos. **TODOS** os métodos são **ABSTRATOS**.



NÃO permite a definição de atributos. **APENAS** constantes estáticas são permitidas.

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

```
interface Guia {  
    guia(): void;  
}
```

```
interface Cacador {  
    rastreia(): void;  
}
```

```
class Cachorro extends Mamifero implements Guia, Cacador {  
    raca: string;  
  
    constructor(cor: string, raca: string) {  
        super(cor)  
        this.raca = raca;  
    }  
  
    // métodos  
  
    rastreia(): void {  
        // implementação do método  
    }  
  
    guia(): void {  
        // implementação do método  
    }  
}
```

DESAFIO

COMBATE

RPG



OBRIGADO!

FEEDBACK

ACESSE

WWW.MENTI.COM

INSIRA O CÓDIGO

6363 6390



ou use o QR code