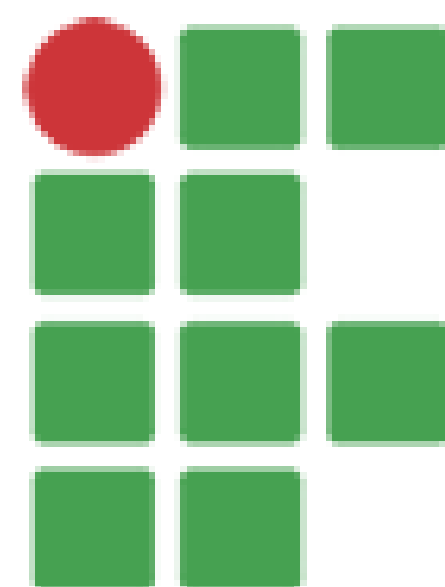




**INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
Rio de Janeiro

Disciplina

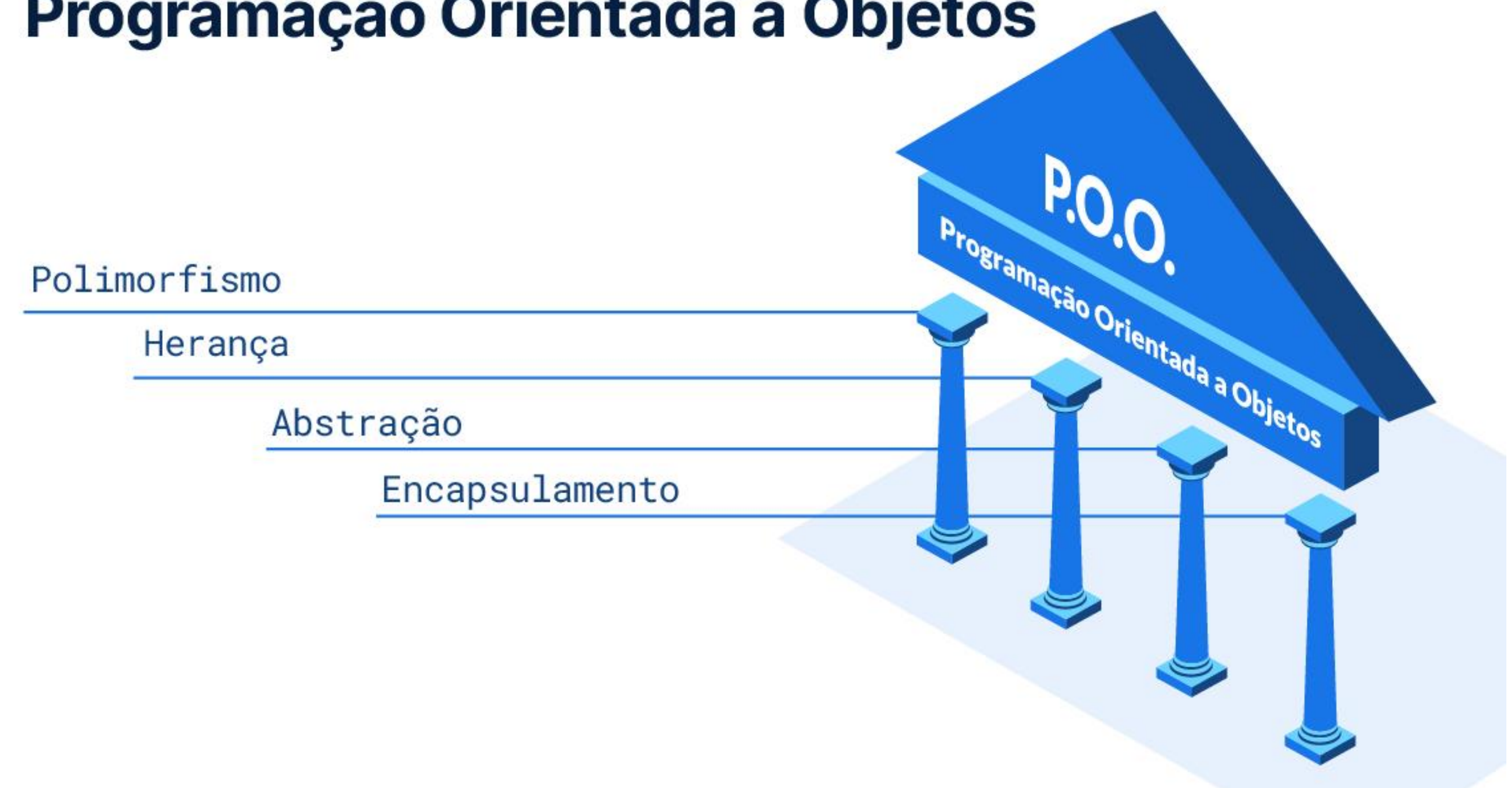
Modelagem de Domínio e
Conceitos de Orientação a Objetos

PROF. FERNANDO TAMBERLINI ALVES

Paradigma Orientação a Objetos

- ✓ Abstração
- ✓ Encapsulamento
- ✓ Herança
- ✓ Polimorfismo

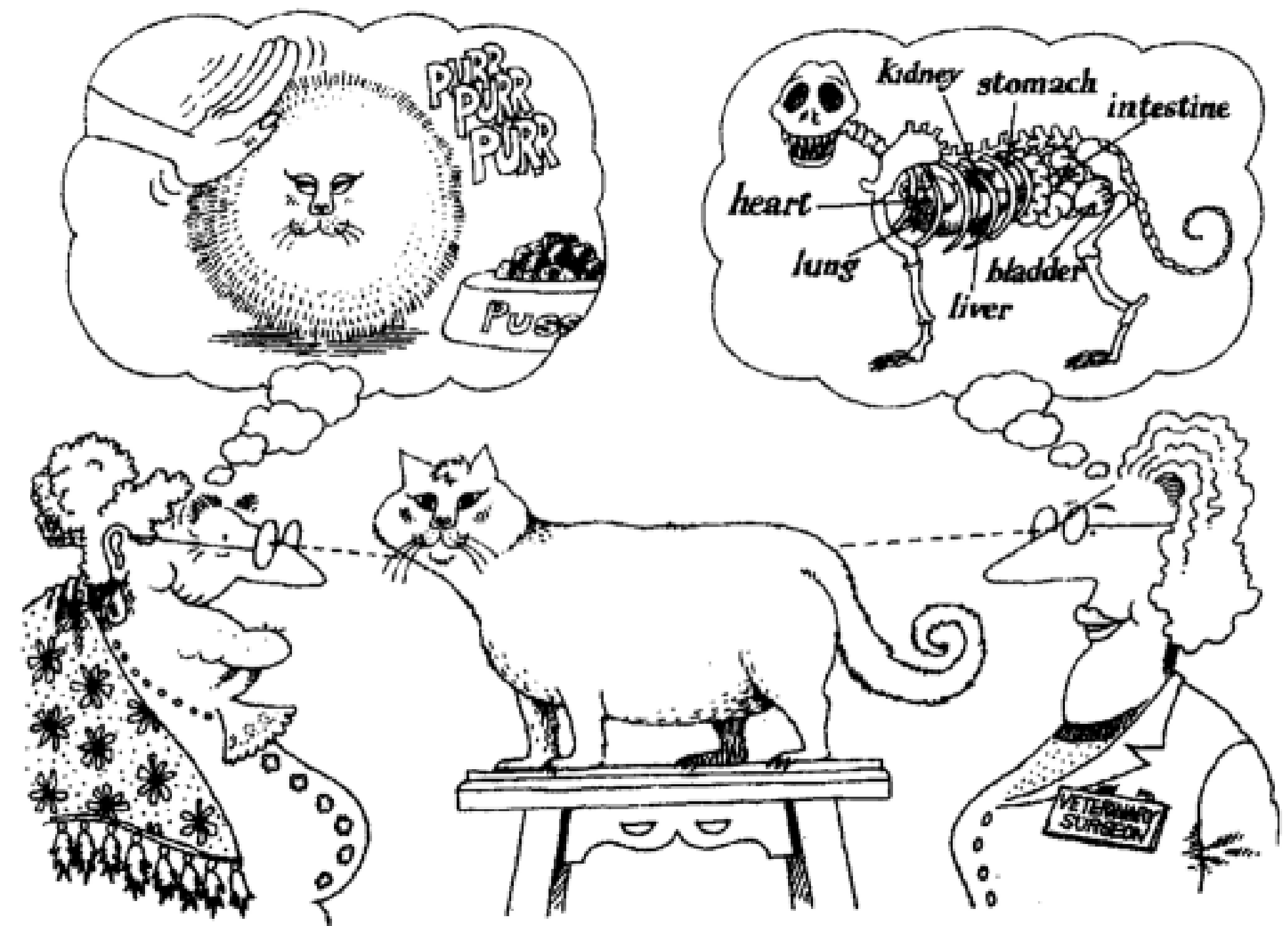
Programação Orientada a Objetos



Paradigma Orientação a Objetos

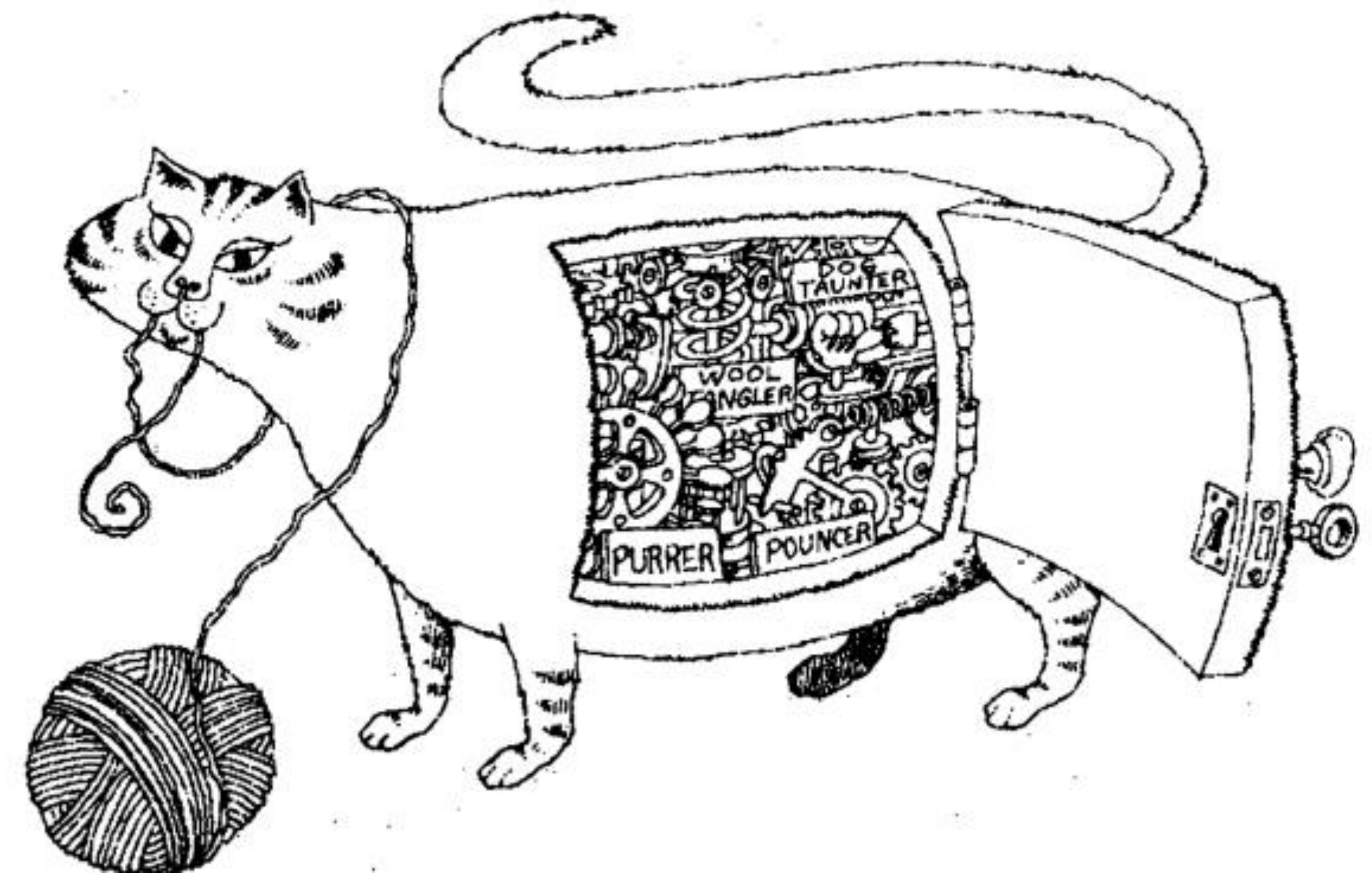
✓ Abstração

- É o princípio de ocultar detalhes complexos de implementação, expondo apenas uma interface simplificada e relevante para o usuário, permitindo que ele interaja com objetos sem se preocupar com suas complexidades internas



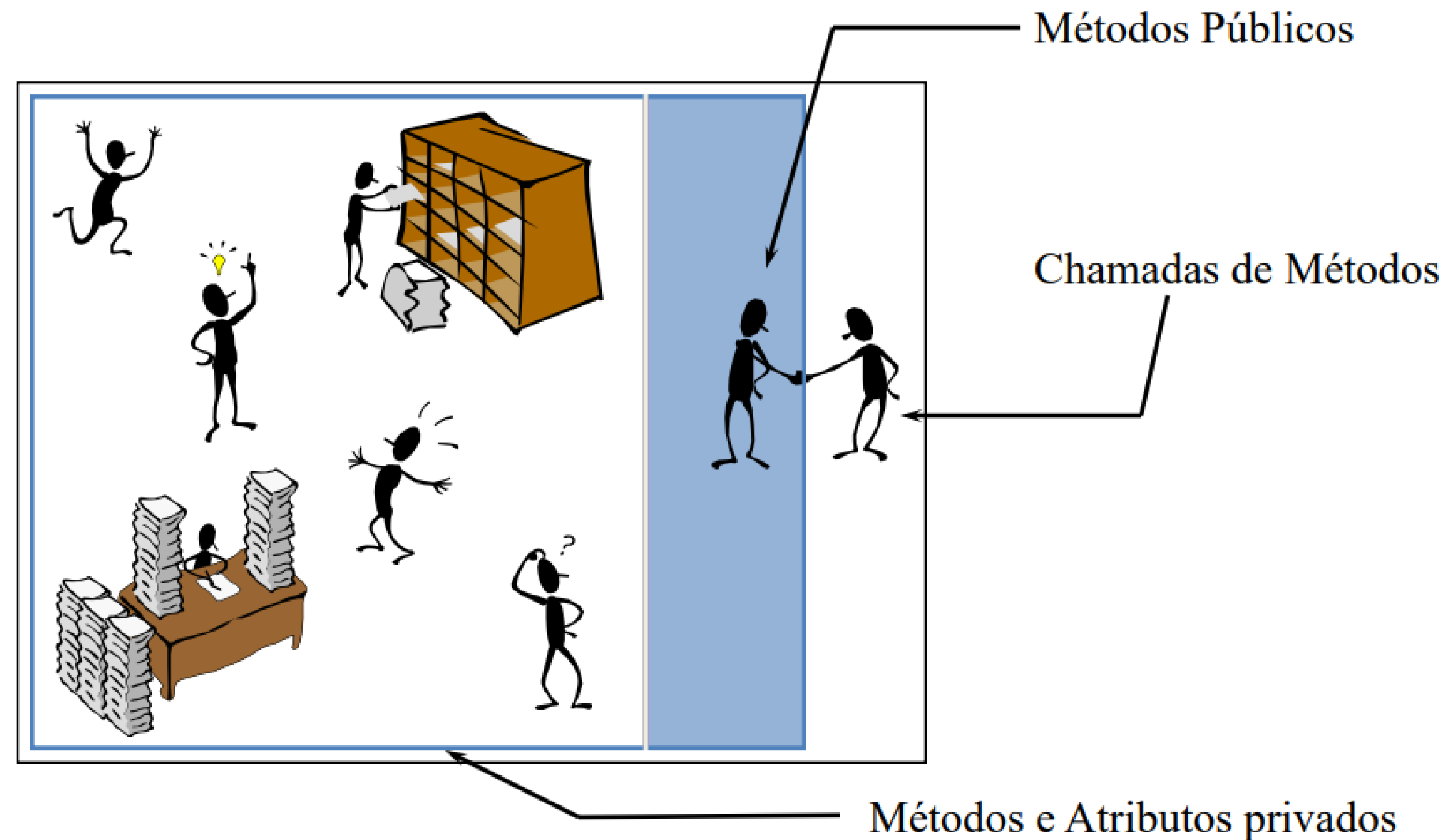
✓ Encapsulamento

- É o princípio de restringir o acesso direto aos dados de um objeto, permitindo que eles sejam manipulados apenas por meio de métodos definidos, protegendo a integridade do estado interno do objeto.



Paradigma Orientação a Objetos

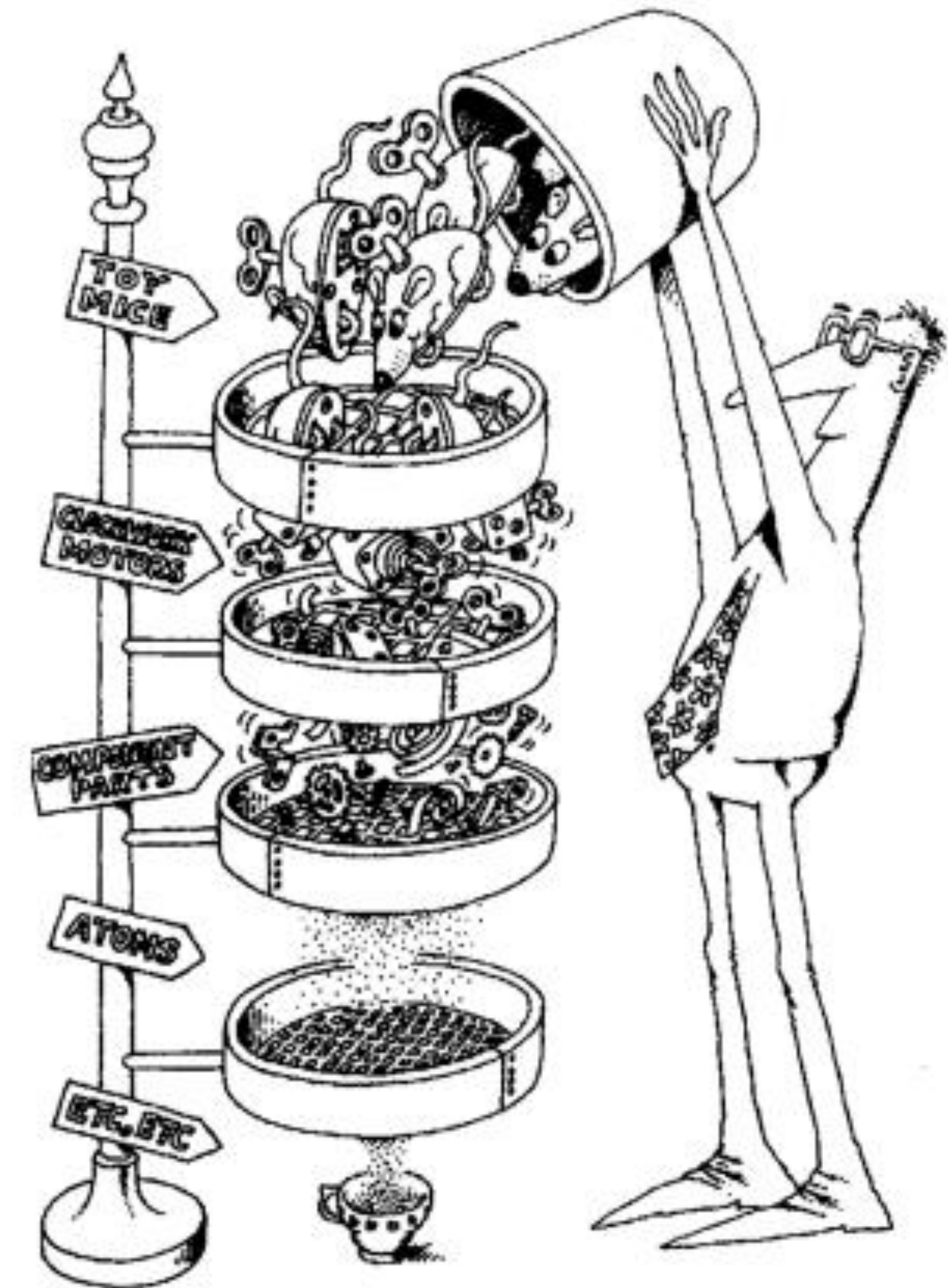
✓ Encapsulamento



Paradigma Orientação a Objetos

✓ Herança

- É o mecanismo pelo qual uma classe (subclasse) pode herdar atributos e métodos de outra classe (superclasse), permitindo reutilização de código e estabelecimento de hierarquias entre classes.

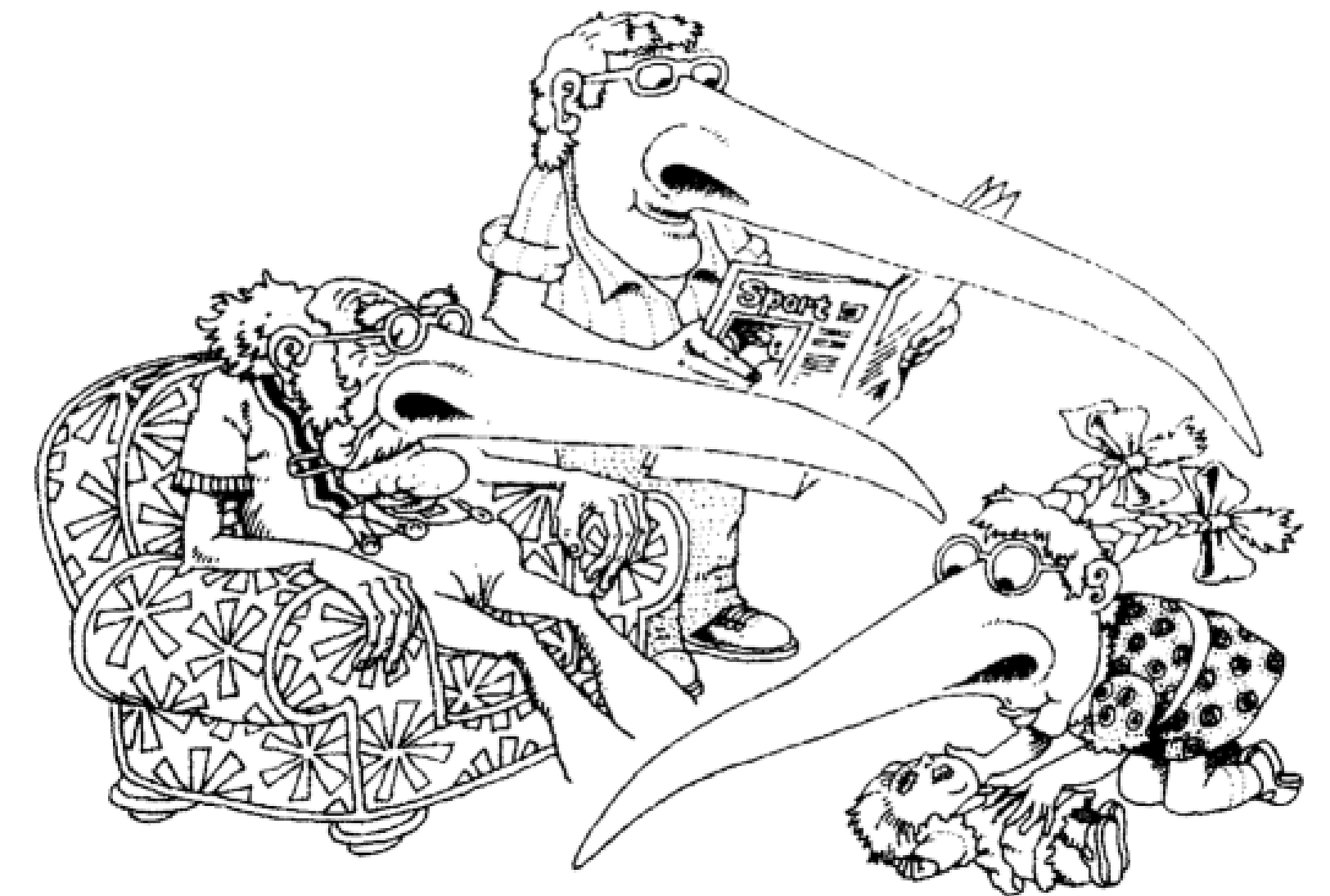


Paradigma Orientação a Objetos

✓ Herança

Para viabilizar a hierarquia entre objetos, as classes são organizadas em estruturas Hierárquicas

- A classe que forneceu os elementos herdados é chamada de superclasse
- A classe herdeira é chamada de subclasse
- A subclasse pode herdar os métodos e atributos de suas superclasses
- A subclasse pode definir novos atributos e métodos específicos



✓ Polimorfismo

Uma subclasse pode redefinir (sobrescrever) um método herdado

- Este mecanismo é chamado de polimorfismo
- O polimorfismo se realiza através da recodificação de um ou mais métodos herdados por uma subclasse
- Em tempo de execução, o Compilador/Interpretador saberá qual implementação deve ser usada

Paradigma Orientação a Objetos

✓ Polimorfismo

O polimorfismo é a capacidade de objetos de diferentes classes responderem ao mesmo método de formas distintas. Isso permite tratar objetos de classes diferentes de maneira uniforme, promovendo flexibilidade e extensibilidade no código.



Paradigma Orientação a Objetos

✓ Encapsulamento

- A partir da versão ES2020 do Javascript foi possível declarar atributos privados incluindo “#” antes do nome do atributo na declaração das classes.

```
cobra > JS alimento.js > ...
1  class Alimento {
2      arqImagem='alimento.png';
3      #tamanho;
4      #valor;
5      #x;
6      #y;
7      #imagem;
8      constructor(valor,tamanho){
9          this.#tamanho = tamanho;
10         this.#valor = valor;
11         this.#x = Math.round(this.#tamanho+(tela.largura-2*this.#tamanho)*Math.random())
12         this.#y = Math.round(this.#tamanho+placar.altura+(tela.altura-2*this.#tamanho)*Math.random())
13         this.#imagem = new Image();
14         this.#imagem.src = this.arqImagem;
15     }
16     desenhar(){
17         canvasCtx.drawImage(this.#imagem, this.#x, this.#y, this.#tamanho, this.#tamanho)
18     }
19 }
```

Paradigma Orientação a Objetos

✓ Herança

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia.

```
cobra > JS objetoJogo.js > ...
1  class ObjetoJogo{
2      constructor(args){
3          this.tamanho = 20;
4          this.x = Math.round(this.tamanho+(tela.largura-2*this.tamanho)*Math.random());
5          this.y = Math.round(placar.altura+(tela.altura-2*this.tamanho)*Math.random());
6          if (args.length >= 1) this.tamanho = args[0];
7          if (args.length >= 2) this.x = args[1];
8          if (args.length >= 3) this.y = args[2];
9      }
10     teveColisao(obj){
11         const distX = this.x - obj.x;
12         const distY = this.y - obj.y;
13         if (((distX >= 0) && (Math.abs(distX) < obj.tamanho)
14             || (distX < 0) && (Math.abs(distX) < this.tamanho))
15             &&
16             ((distY >= 0) && (Math.abs(distY) < obj.tamanho)
17             || (distY < 0) && (Math.abs(distY) < this.tamanho)))
18             return true;
19         return false;
20     }
21 }
```


Paradigma Orientação a Objetos

✓ Herança

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia.

```
cobra > JS alimento.js > ...
1  class Alimento extends ObjetoJogo {
2      |   arqImagem='alimento.png';
3      |   #imagem;
4      |   constructor(valor,...args){
5      |       |   super(args);
6      |       |   this.valor = valor;
7      |       |   this.#imagem = new Image();
8      |       |   this.#imagem.src = this.arqImagem;
9      |       |   }
10     |   desenhar(){
11     |       |   canvasCtx.drawImage(this.#imagem, this.x, this.y, this.tamanho, this.tamanho);
12     |       |   }
13     |   }
```

Paradigma Orientação a Objetos

✓ Polimorfismo

Objetos podem responder de maneira diferente aos mesmos métodos, permitindo que um programa seja mais flexível e genérico.

```
cobra > JS jogo.js > document.addEventListener("keydown") callback
```

```
1 function jogar () {
2   tela.desenhar()
3   placar.desenhar()
4   cobra.desenhar()
5   alimento.desenhar()
6   cobra.mover()
7   if (alimento.teveColisao(cobra)) {
8     placar.qtdPontos += alimento.valor;
9     alimento = new Alimento(alimento.valor++);
10  }
11  if (cobra.vida > 0)
12    requestAnimationFrame(jogar)
13  else {
14    placar.nomeJogo = "Fim de Jogo";
15    placar.desenhar()
16  }
17 }
```

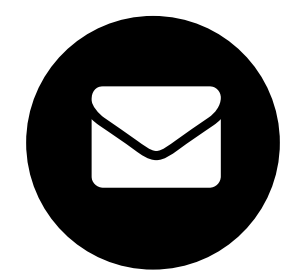
```
cobra > JS jogo.js > document.addEventListener("keydown") callback
```

```
18 let alimento = new Alimento(1)
19 //let alimento = new Alimento(1,30)
20 //let alimento = new Alimento(1,30,400)
21 //let alimento = new Alimento(1,30,460)
22 jogar()
23 document.addEventListener("keydown", (evento) => {
24   if (evento.key == 6) cobra.direcao = 0;
25   if (evento.key == 2) cobra.direcao = 90;
26   if (evento.key == 4) cobra.direcao = 180;
27   if (evento.key == 8) cobra.direcao = 270;
28   // console.log("Tecla Pressionada: " + evento.key);
29 });
```

Dúvidas?



Dúvidas?



fernando.alves@ifrj.edu.br



ftamberlini.dev.br