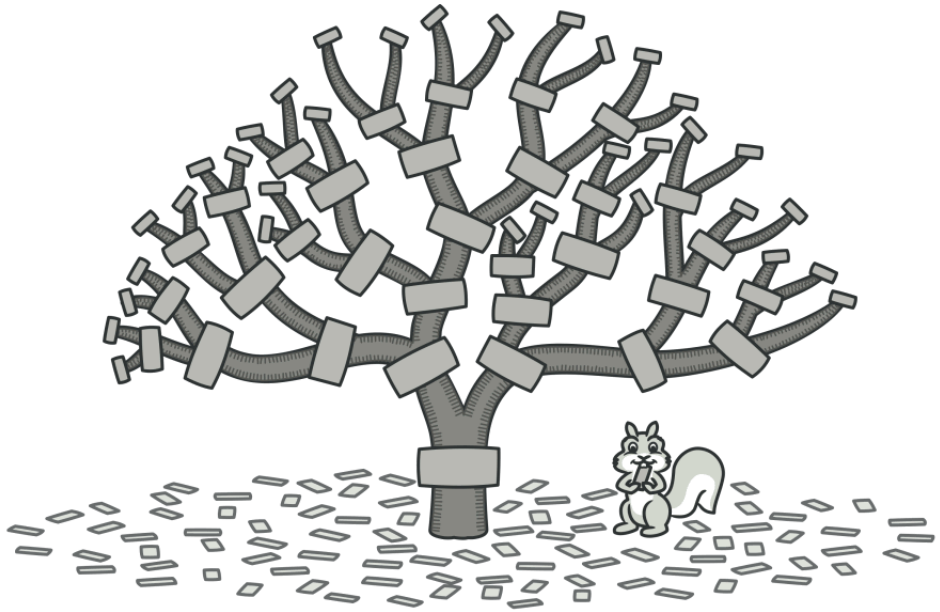
TÓPICO 11 - COMPOSITE

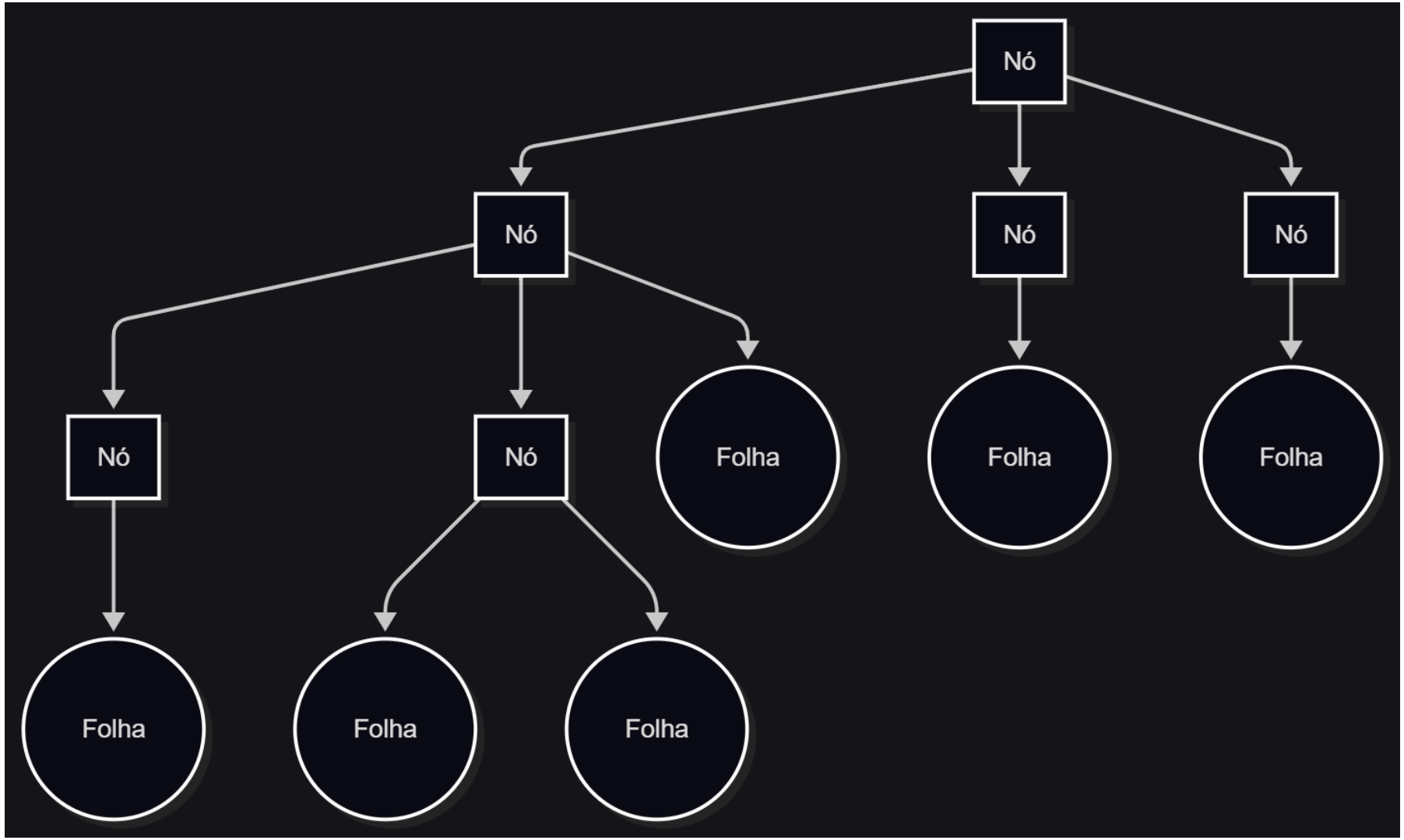
Design Patterns - Professor Ramon Venson - SATC 2026.1

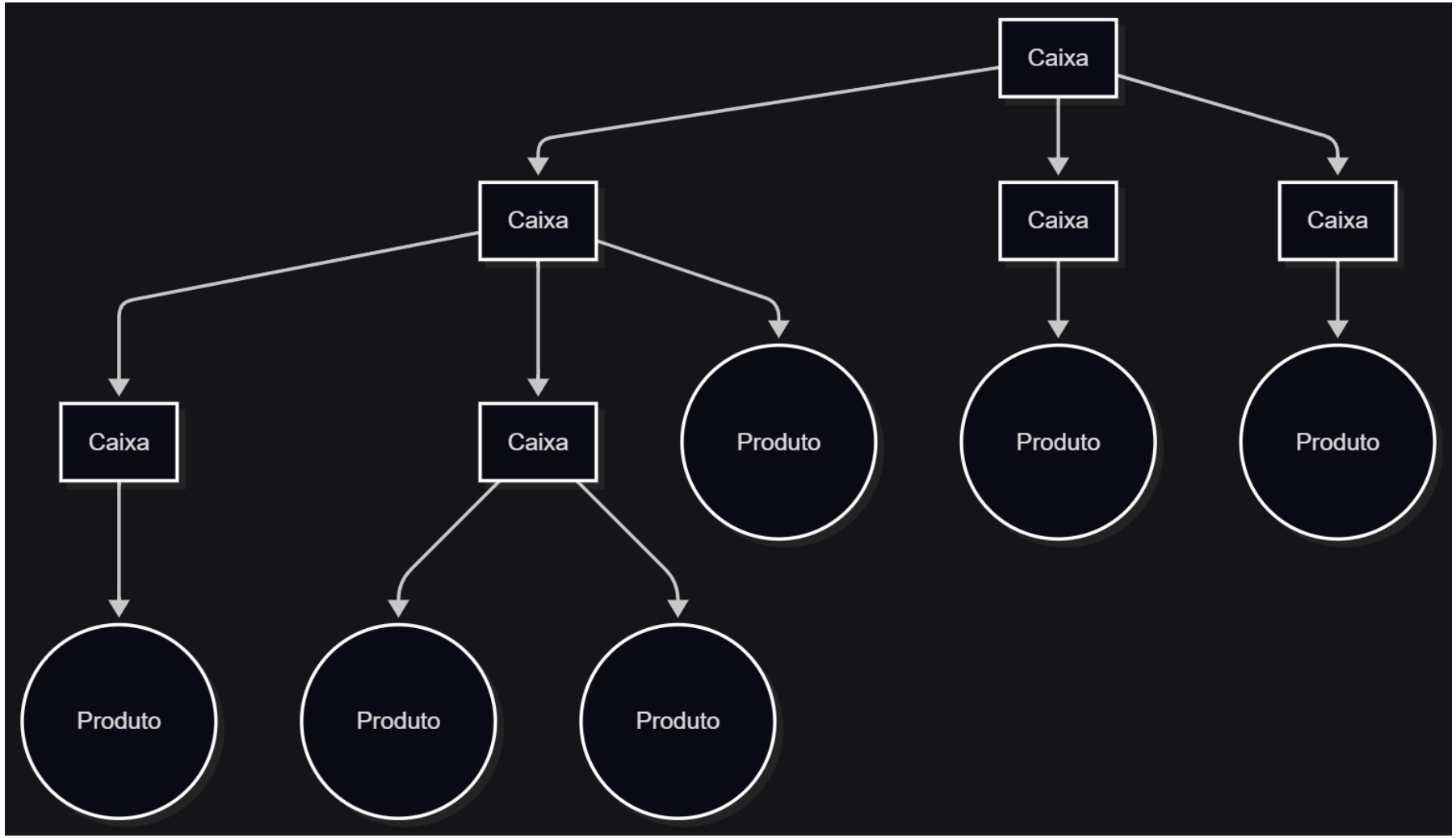
Motivação

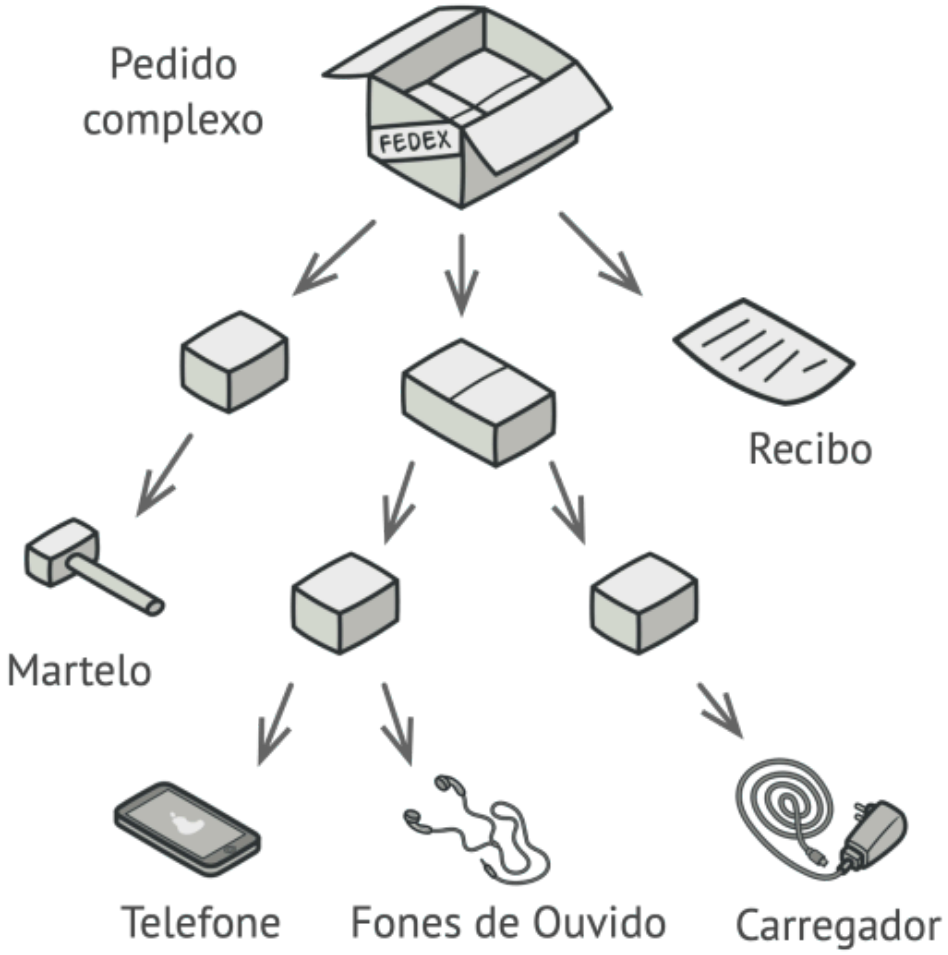
Em alguns casos, temos objetos que podem conter outros objetos do mesmo tipo (recursividade).

O padrão Composite é uma solução para tratar objetos individuais e composições de objetos de forma uniforme.




Nó
Nó
Nó
Nó
Nó
Nó
Folha
Folha
Folha
Folha
Folha
Folha


Caixa
Caixa
Caixa
Caixa
Caixa
Caixa
Produto
Produto
Produto
Produto
Produto
Produto



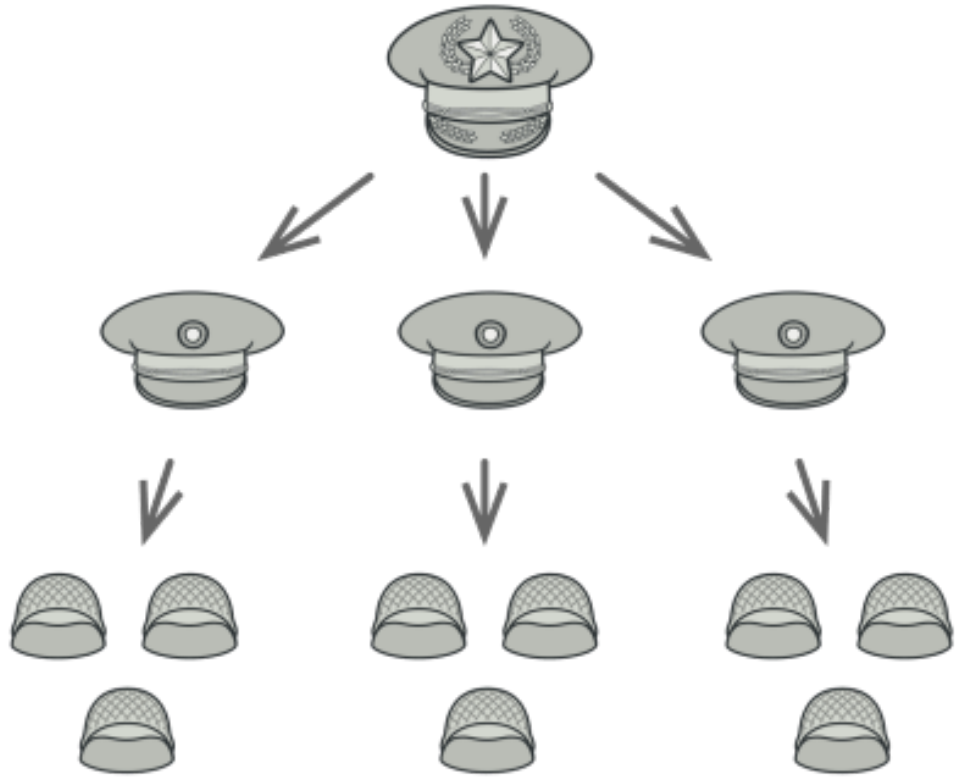
Problema

Imagine um sistema complexo onde uma caixa de pedidos pode conter tanto produtos quanto outras caixas.

Cliente acaba usando vários `if` e `instanceof` ou a recursão fica espalhada pelo sistema

Conceito principal

- **Composite** é um padrão estrutural
- Ele usa uma interface comum
- Folhas e composites compartilham o mesmo contrato
- O cliente trata ambos da mesma forma
- A arvore responde recursivamente

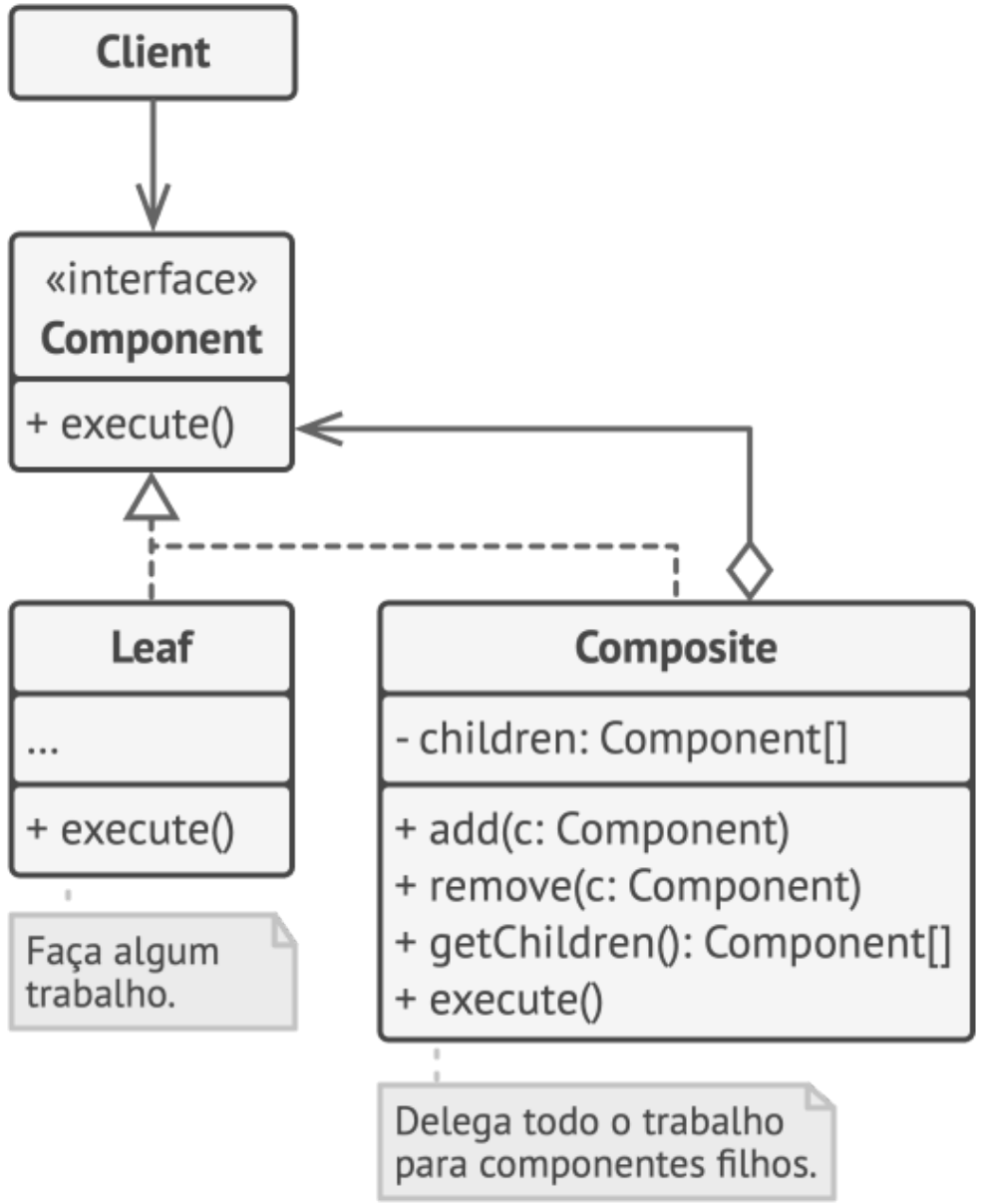


Como funciona

- A folha executa a operação diretamente
- O composite guarda filhos da mesma abstração
- Ao receber chamada, delega aos filhos
- Depois combina os resultados obtidos
- O comportamento percorre toda a árvore

Estrutura / Componentes

- **Componente** define operação comum
- **Folha** representa elemento indivisível
- **Composite** agrega filhos
- Filhos podem ser folhas ou composites
- Cliente depende apenas da abstração



Exemplo

- Sistema de cursos online
- Uma `Aula` é elemento simples
- Uma `Unidade` agrupa aulas e outras unidades
- Ambos calculam carga horária
- Cliente monta a estrutura como árvore

Código: componente e folha

```
interface ComponenteCurso {
    int calcularCargaHorária();
}

class Aula implements ComponenteCurso {
    private int horas;

    public int calcularCargaHorária() {
        return horas;
    }
}
```

Código: composite

```
class Unidade implements ComponenteCurso {
    private List<ComponenteCurso> componentes = new ArrayList<>();

    public void adicionar(ComponenteCurso componente) {
        componentes.add(componente);
    }

    public int calcularCargaHorária() {
        int total = 0;
        for (ComponenteCurso componente : componentes) {
            total += componente.calcularCargaHorária();
        }
        return total;
    }
}
```

Quando usar

- O modelo central é hierárquico
- Há estrutura de árvore ou subárvore
- Cliente precisa uniformidade de tratamento
- Operações recursivas são frequentes
- O domínio tem partes e todo bem definidos

Quando não usar

- O problema não é hierárquico
- Não existe recursão relevante
- Há poucos tipos e pouca variação
- Interface comum ficaria artificial demais
- O custo extra de modelagem não compensa

Vantagens

- Simplifica o código cliente
- Favorece polimorfismo e recursão
- Facilita extensão da estrutura
- Reduz condicionais por tipo concreto
- Representa bem domínios em árvore

Desvantagens

- Pode generalizar demais a interface
- Aumenta número de classes
- Nem toda coleção precisa de Composite
- Validações podem ficar mais delicadas
- Pode confundir iniciantes no início