



**Ministério da Educação  
Instituto Federal de Mato Grosso  
Campus Cuiabá – Cel. Octayde Jorge da Silva  
Departamento de Área de Computação**



# **Design Pattern: Composite**

**Discente: Náira de Moura e Souza  
Docente: João Paulo Preti**

IFMT - Campus Cuiabá

Cuiabá, MT – 05/09/2022



# Agenda

- 1. Sobre o composite**
- 2. Estrutura**
- 3. Benefícios**
- 4. Malefícios**
- 5. Implementação**



# 1. Sobre o composite

- É um padrão de composição;
- É um padrão estrutural;
- Possui uma estrutura hierárquica (árvore);
- Trata estruturas complexas como estruturas uniformes;
- Prioriza a composição ao invés da herança



## 2. Estrutura

- Possui interface em comum para “Composite” e “Leaf”;
- Objetos do tipo “Composite” são objetos que têm filhos e delegam o trabalho para eles;
- Objetos do tipo “Leaf” são os que realmente fazem o trabalho



### 3. Benefícios

- **Facilidade em criar objetos complexos por composição;**
- **Facilidade em criar uma hierarquia de objetos;**
- **Facilidade em usar polimorfismo e recursão;**
- **Facilidade em adicionar novos elementos na estrutura**



## 4. Malefícios

- **Risco de ocorrer uma quebra do princípio da segregação de interface (ISP), pois objetos do tipo “Leaf” tendem a ter métodos que não usam ou não fazem nada**



## 5. Implementação

```
class Component(ABC):  
  
    @property  
    def parent(self) -> Component:  
        return self._parent  
  
    @parent.setter  
    def parent(self, parent: Component):  
        self._parent = parent  
  
    def add(self, component: Component) -> None:  
        pass  
  
    def remove(self, component: Component) -> None:  
        pass  
  
    def is_composite(self) -> bool:  
        return False  
  
    @abstractmethod  
    def operation(self) -> str:  
        pass
```



## 5. Implementação

```
class Folha(Component):  
    def operation(self) -> str:  
        return "Folha"
```





## 5. Implementação

```
class Composite(Component):  
  
    def __init__(self) -> None:  
        self._children: List[Component] = []  
  
    def add(self, component: Component) -> None:  
        self._children.append(component)  
        component.parent = self  
  
    def remove(self, component: Component) -> None:  
        self._children.remove(component)  
        component.parent = None  
  
    def is_composite(self) -> bool:  
        return True  
  
    def operation(self) -> str:  
  
        results = []  
        for child in self._children:  
            results.append(child.operation())  
        return f"Galho({'+'.join(results)})"  
  
def code(component: Component) -> None:  
  
    print(f"{component.operation()}", end="")  
  
def code2(component1: Component, component2: Component) -> None:  
  
    if component1.is_composite():  
        component1.add(component2)  
  
    print(f"{component1.operation()}", end="")
```



**Obrigada!**

**Naíra de Moura e Souza**

**Email:** [nairamouras@gmail.com](mailto:nairamouras@gmail.com)