Métodos Avançados de Programação

PADRÕES GOF

Drª. ALANA MORAIS

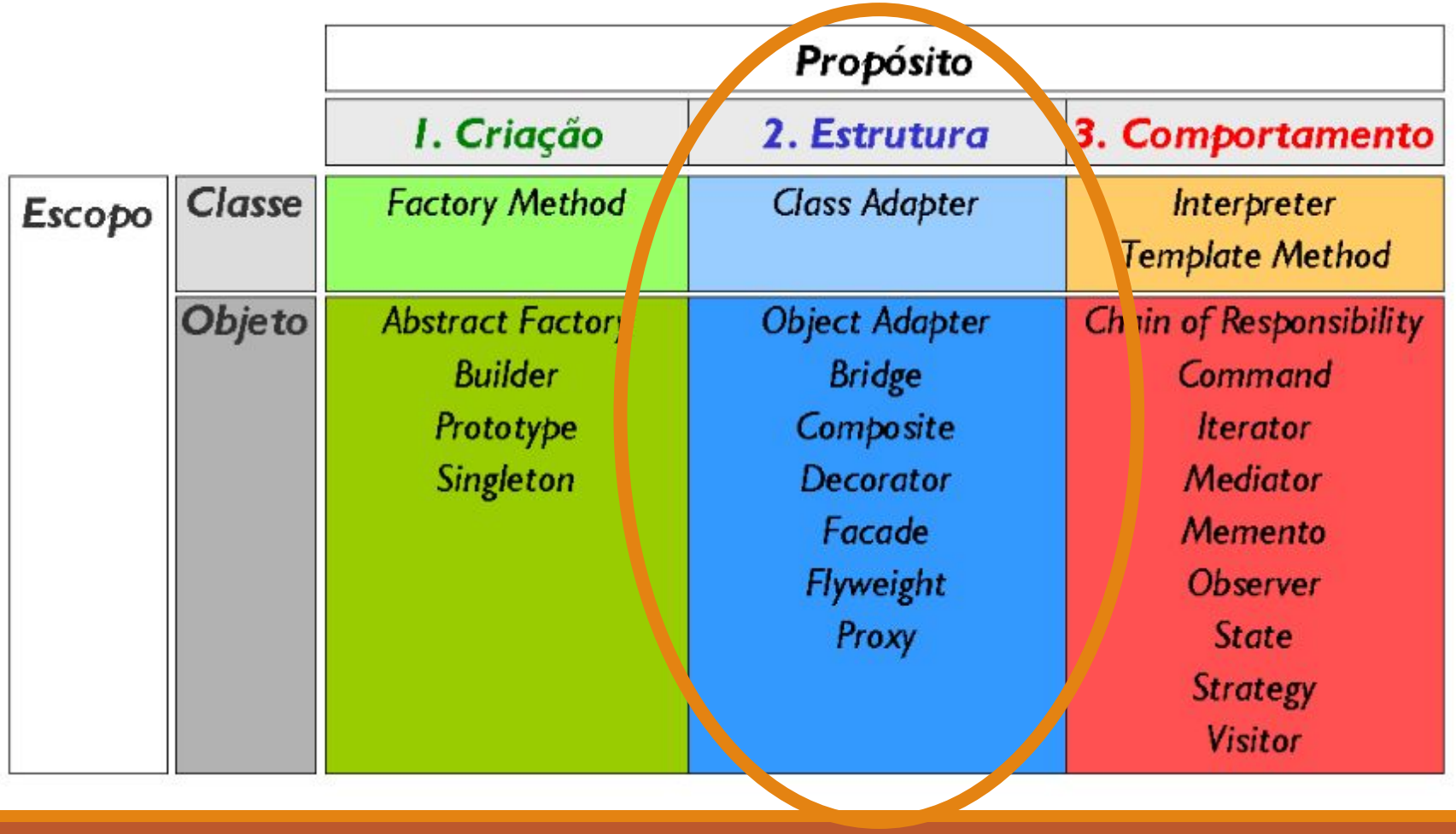
Aula Passada

Finalizamos os padrões criacionais do GOF

- Adapter (Classe e Objeto)
- Bridge
- Façade

Qual a diferença entre um Adapter de Classe e de Objeto?

Padrões GoF – Padrões de Estrutura



		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Composite

"COMPOR OBJETOS EM ESTRUTURAS DE ÁRVORE PARA REPRESENTAR HIERARQUIAS TODO-PARTE. COMPOSITE PERMITE QUE CLIENTES TRATEM OBJETOS INDIVIDUAIS E COMPOSIÇÕES DE OBJETOS DE MANEIRA UNIFORME."

Composite

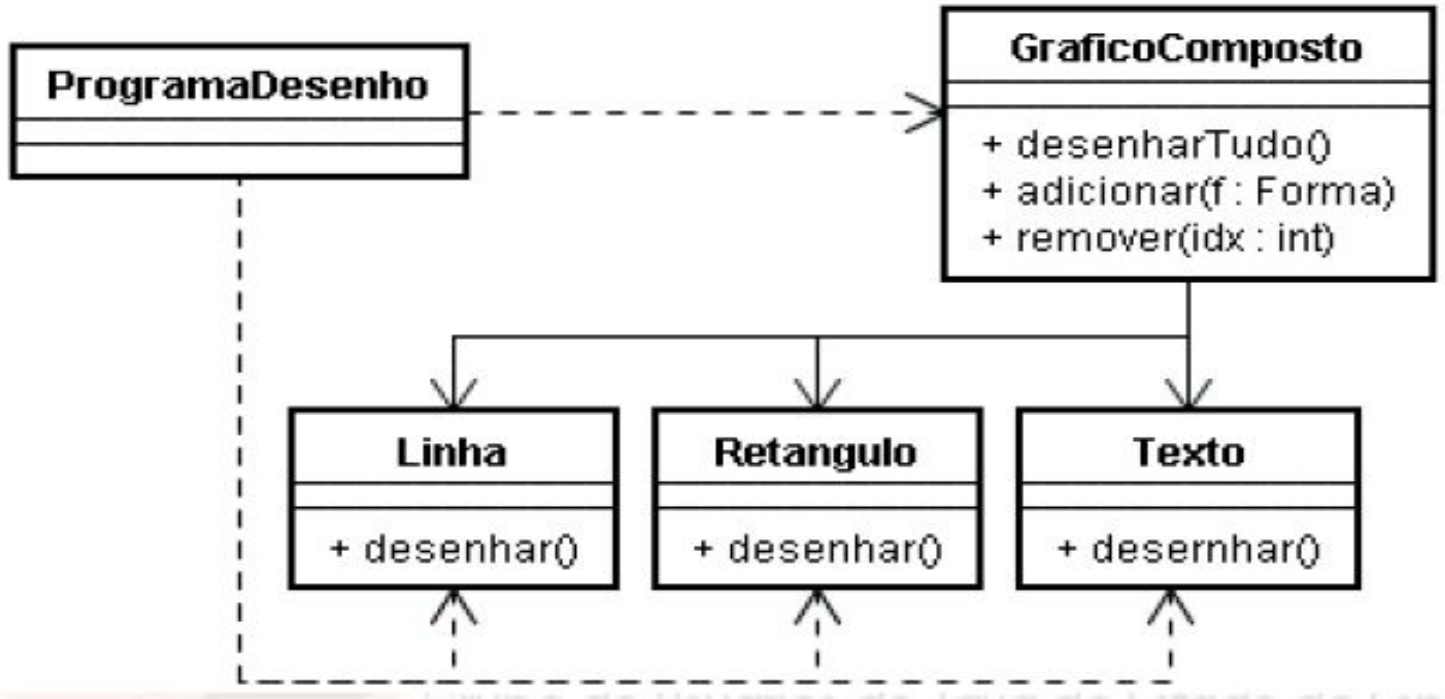
Intenção:

- Compor objetos em estruturas de árvore para representar hierarquias todo-parte.
- Permite que clientes trate objetos individuais e compostos de maneira uniforme.
- O padrão Composite é normalmente utilizado para representar listas recorrentes - ou recursivas - de elementos.

Composite
Problema

Existem gráficos que são compostos de outros gráficos.

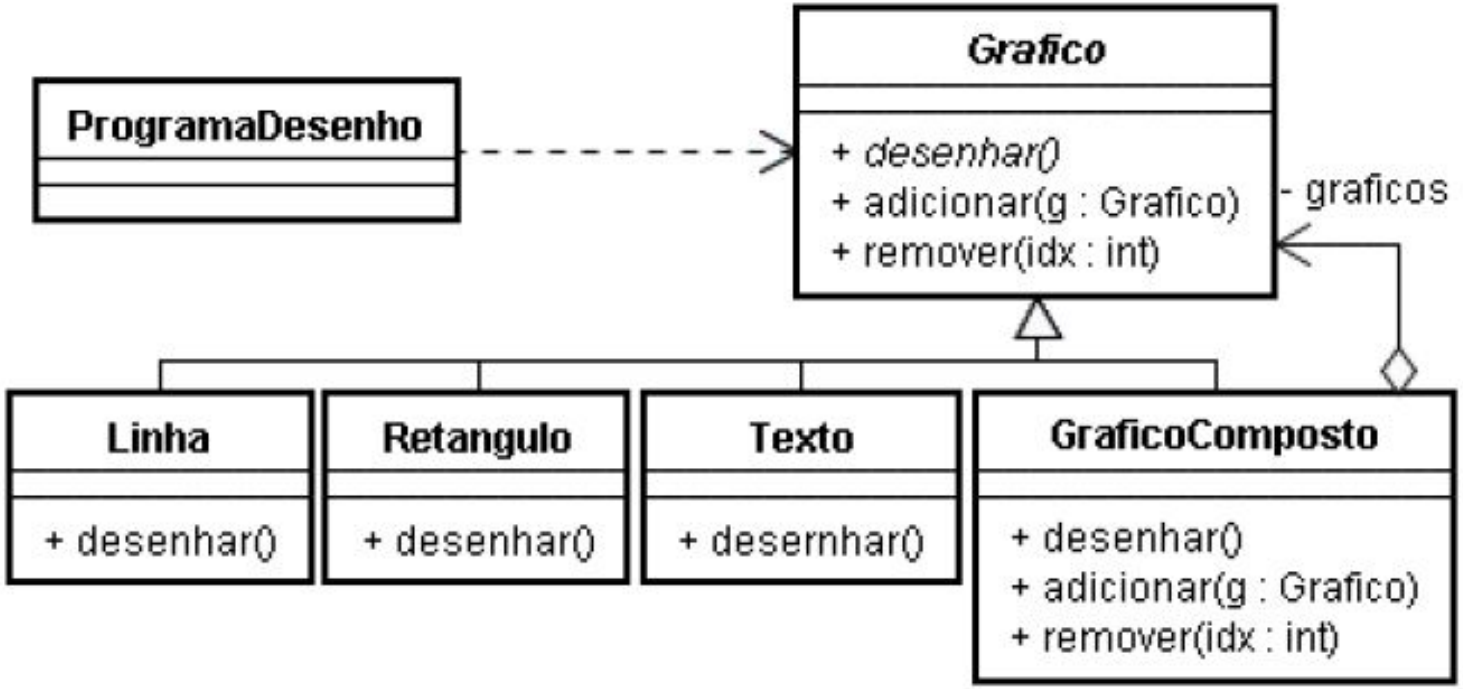
O programa tem que conhecer cada um deles, o que complica o código.



Composite
Solução

A classe abstrata representa tanto gráficos simples quanto compostos;

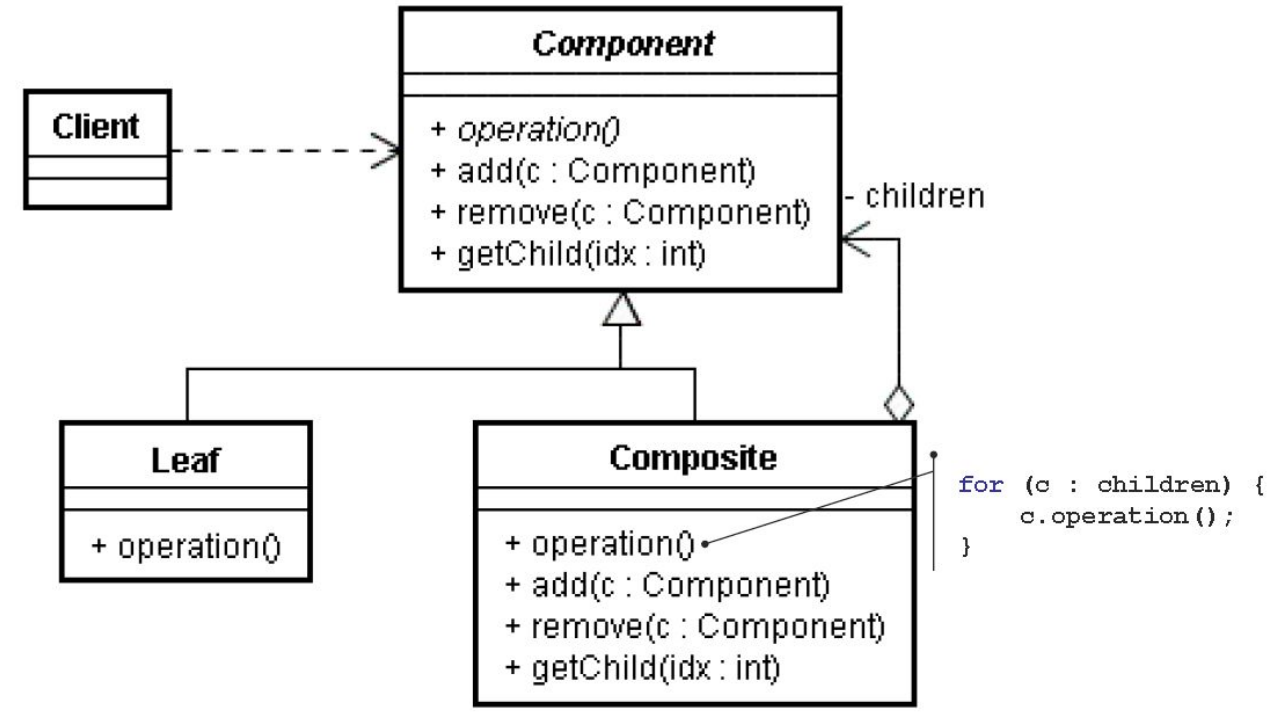
Programa só precisa conhecer gráfico.



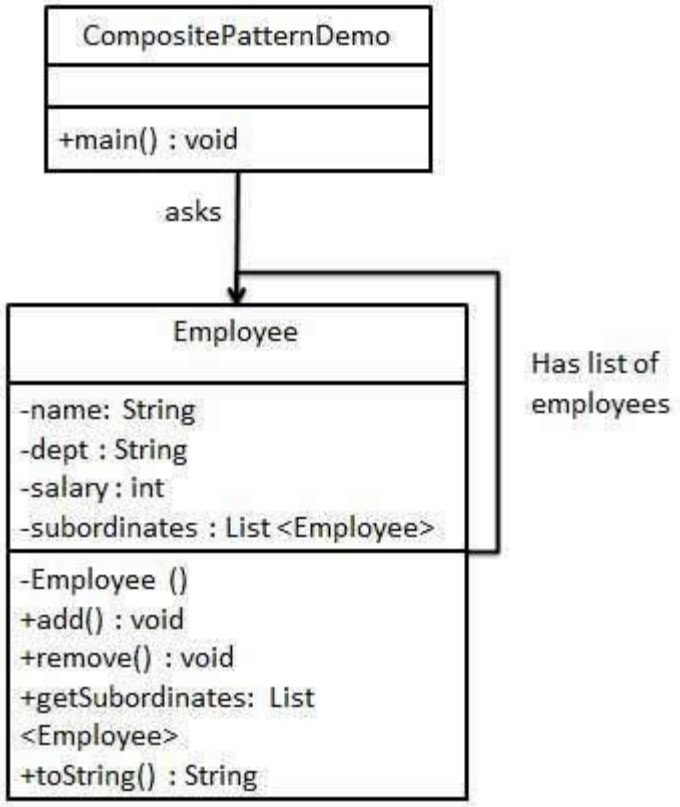
Composite Estrutura

Os nomes genéricos dados às classes abstratas são **Component** (pode ser uma interface) e **Composite**

Os nomes genéricos dados às classes concretas são **Leaf** e **ConcreteComposite** (Classes que herdam a classe Composite)



Composite
Exemplo 2



Composite
Quando usar?

Sempre que houver necessidade de tratar um conjunto como um indivíduo

Funciona melhor se relacionamentos entre os objetos for uma árvore

- Caso o relacionamento contenha ciclos, é preciso tomar precauções adicionais para evitar loops infinitos, já que Composite depende de implementações recursivas
- Há várias estratégias de implementação.

Composite
Vantagens e desvantagens

Define hierarquias todo-parte:

- Objetos podem ser compostos de outros objetos e assim por diante.

Simplifica o cliente:

- Clientes não se preocupam se estão lidando com compostos ou individuais.

Facilita a criação de novos membros:

- Basta estar em conformidade com a interface comum a todos os componentes.

Pode tornar o projeto muito genérico:

- Qualquer componente pode ser criado, não há como usar checagem de tipos para restringir.

Composite
Exercício

Como poderia ser resolvido o problema abaixo?

- É preciso saber quantas pessoas irão participar do congresso
- Participantes podem ser pessoas ou instituição

Decorator

“ANEXAR RESPONSABILIDADES ADICIONAIS A UM OBJETO DINAMICAMENTE. DECORATOR OFERECE UMA ALTERNATIVA FLEXÍVEL AO USO DE HERANÇA PARA ESTENDER UMA FUNCIONALIDADE."

Decorator

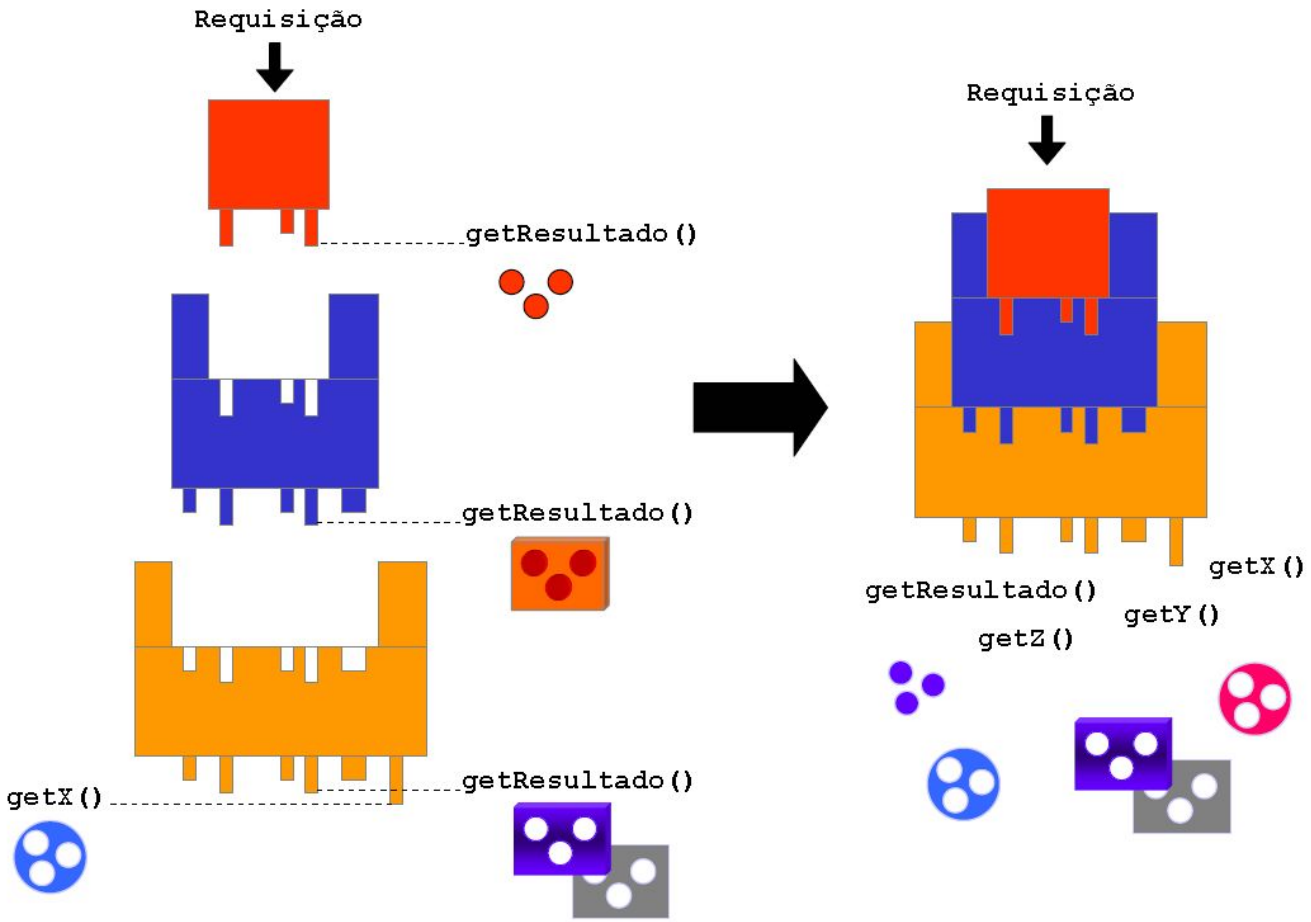
Anexar responsabilidades adicionais a um objeto dinamicamente

Decoradores fornecem uma alternativa flexível em relação a herança para estender funcionalidades

Use o Decorator quando:

- Quiser adicionar responsabilidades a objetos dinamicamente
- Quando a extensão por subclasses é impraticável

Decorator Problema



Decorator Problema

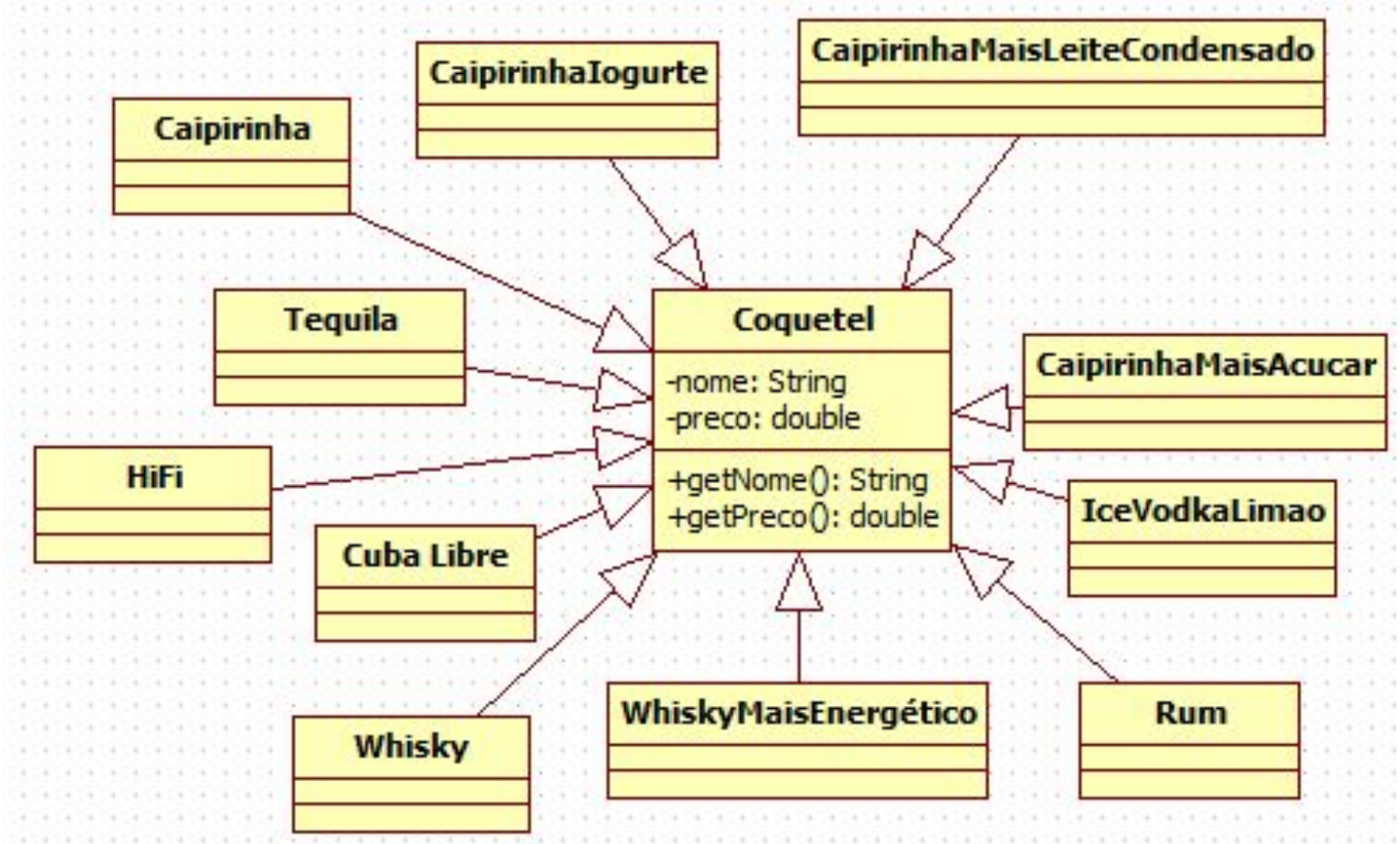
Imagine que você está desenvolvendo um sistema para um bar especializado em coquetéis, onde existem vários tipos de coquetéis que devem ser cadastrados para controlar a venda.

Conjunto de bebidas:	Conjunto de adicionais:
Cachaça Rum Vodka Tequila	Limão Refrigerante Suco Leite condensado Gelo Açúcar

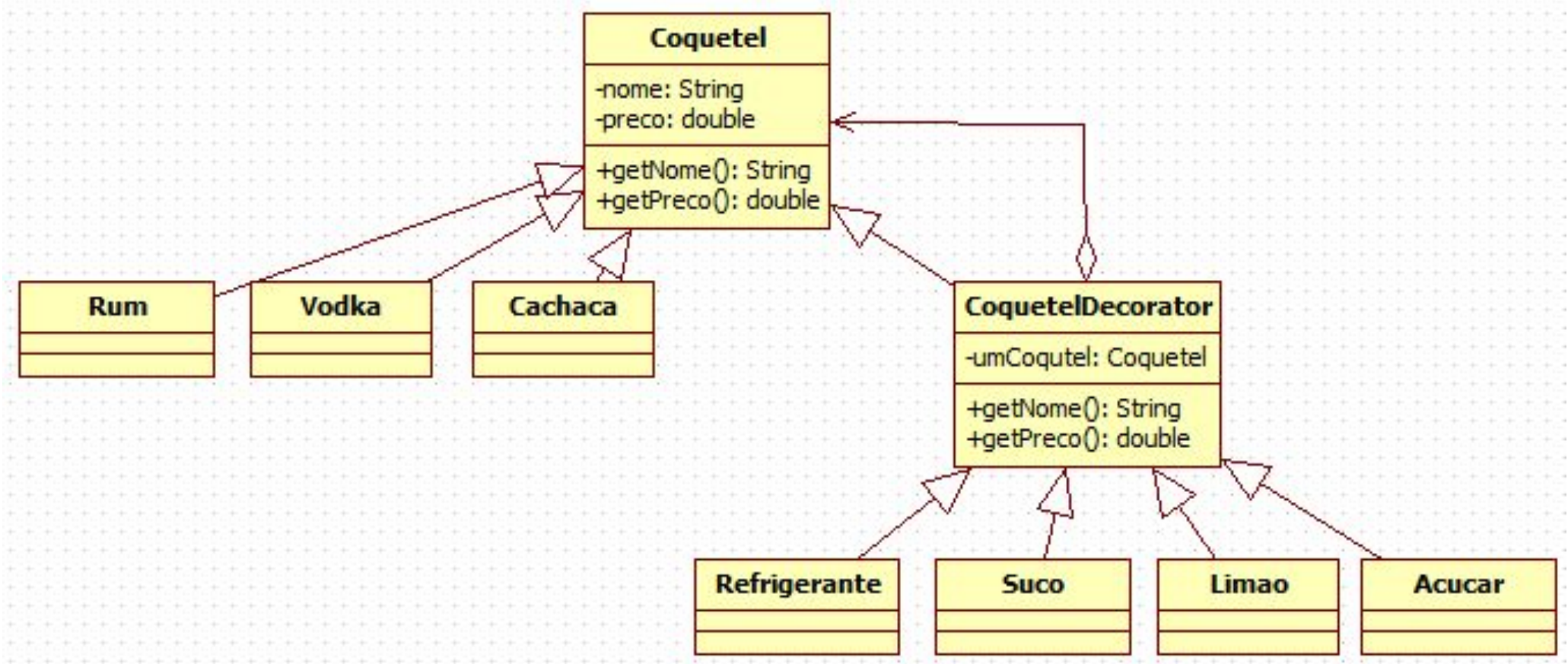
Então, como possíveis coquetéis temos: (a) Vodka + Suco + Gelo + Açúcar, (b) Tequila + Limão + Sal, (c) Cachaça + Leite Condensado + Açúcar + Gelo

E então, como representar isto em um sistema computacional?

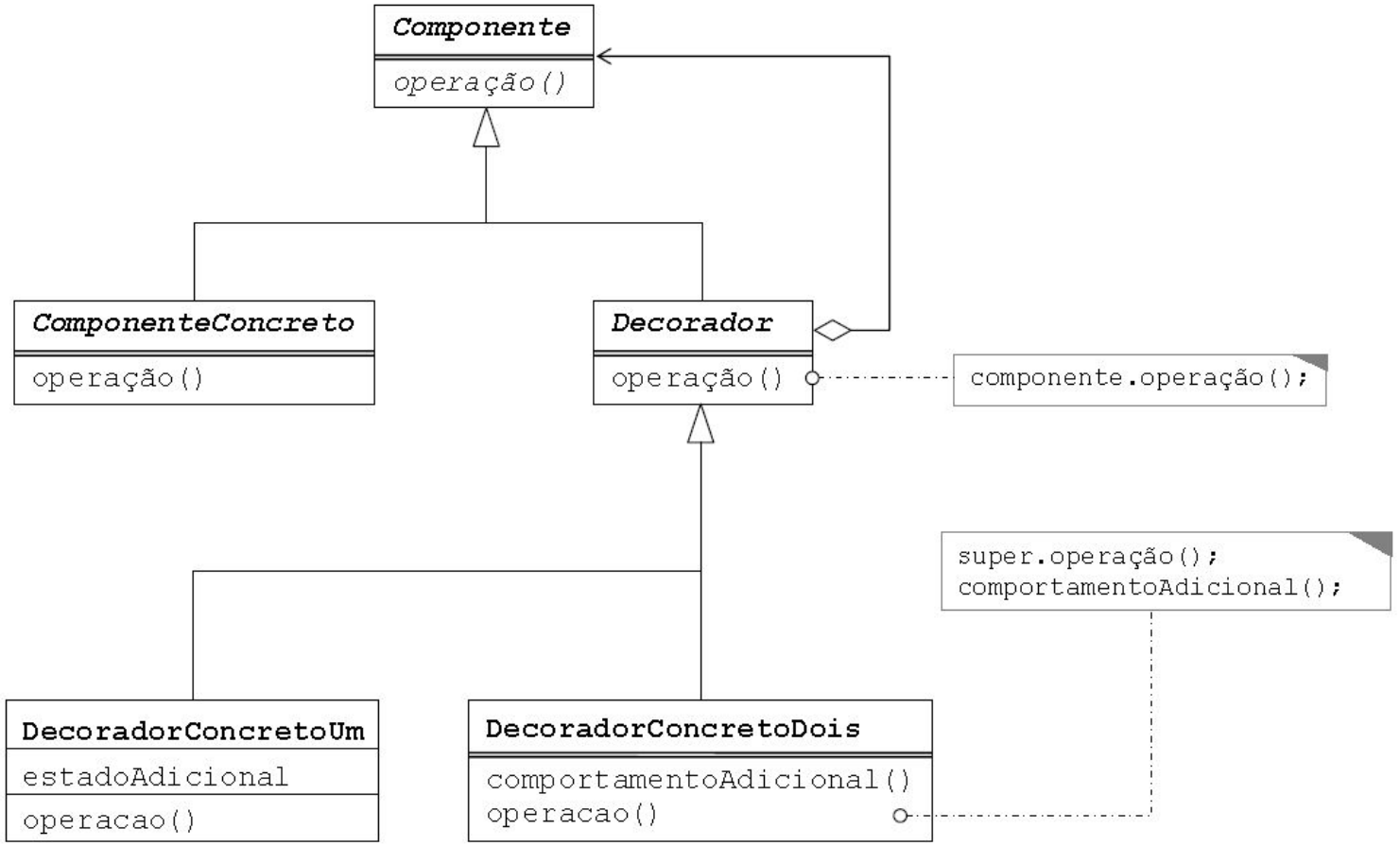
Decorator Problema



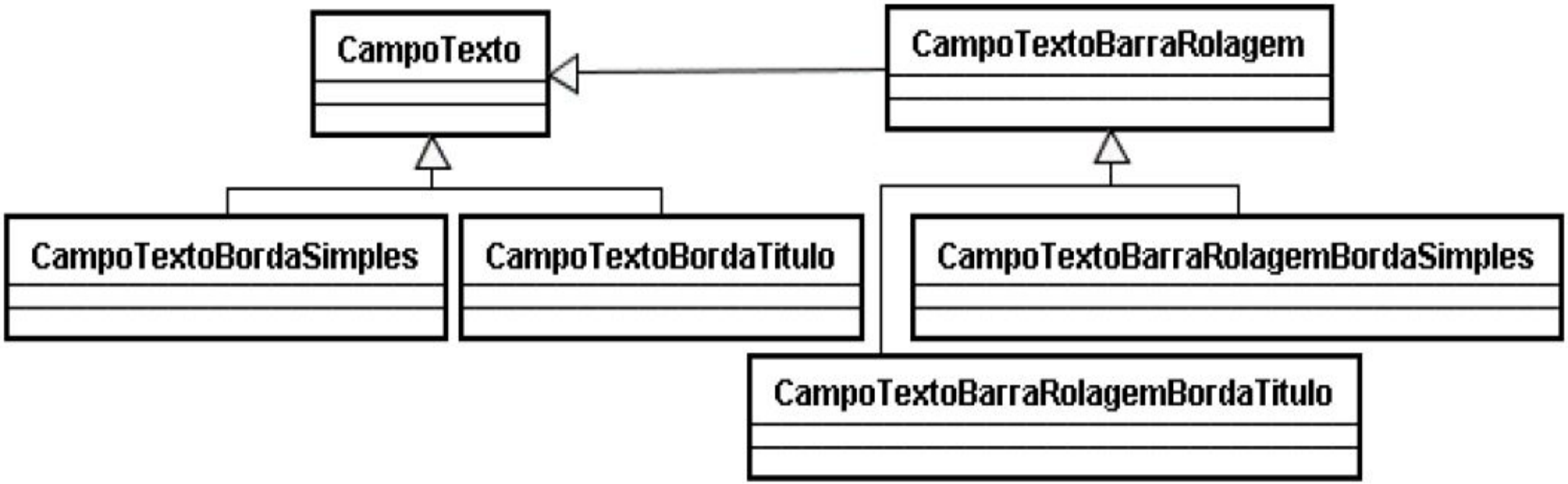
Decorator Problema



Decorator
Estrutura do Decorator

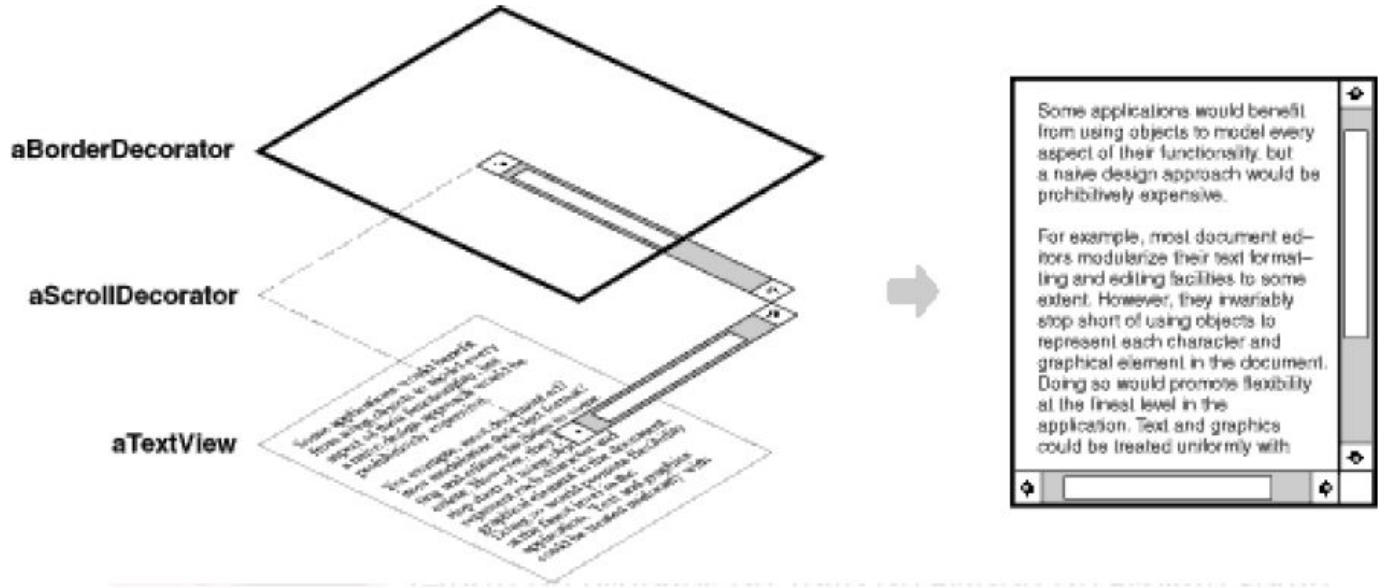


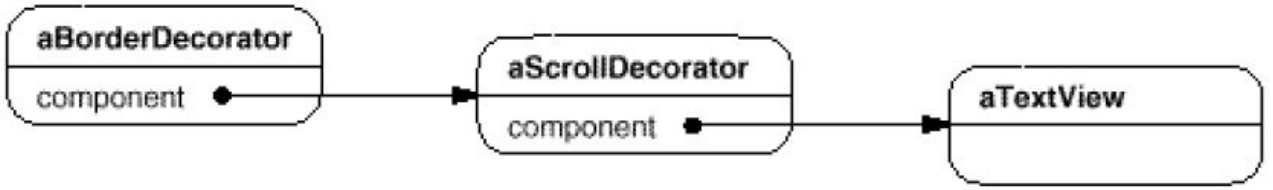
Decorator
Problema



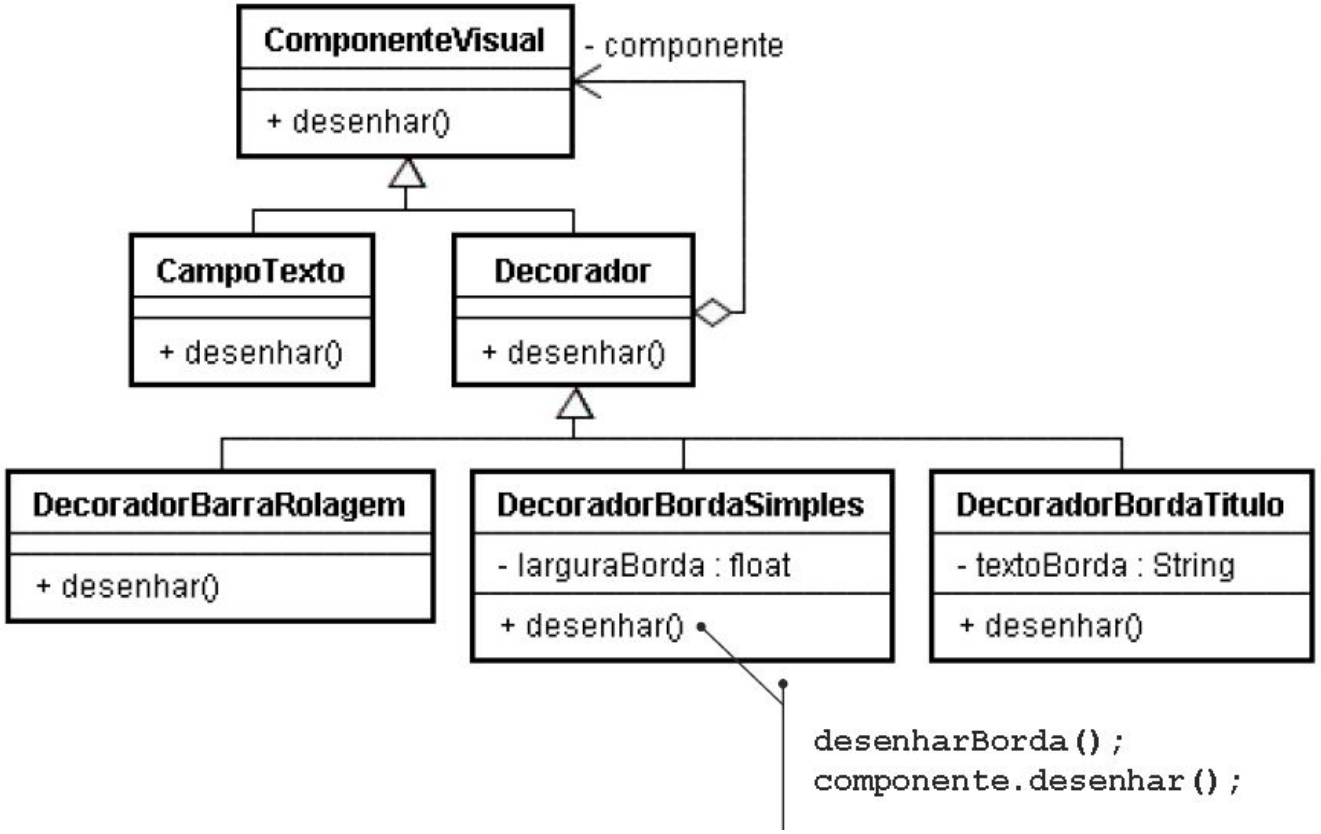
Decorator
Solução (1)

Componentes adicionados por cima de outros, decorando-os.





Decorator
Solução (2)



Decorator
Vantagens e Desvantagens

Mais flexibilidade do que herança:

- Podem ser adicionadas/removidas em tempo de execução;
- Pode adicionar duas vezes a mesma funcionalidade.

O decorador é diferente do componente:

- A identidade do objeto não pode ser usada de forma confiável.

Muitos objetos pequenos:

- □ Um projeto que utiliza Decorator pode vir a ter muitos objetos pequenos e parecidos.

Decorator
Exercício

Implemente a arquitetura apresentada nos slide 21 a 23.

Dúvidas?

ALANAMM.PROF@GMAIL.COM