

**INSTITUTO  
FEDERAL**

Ceará

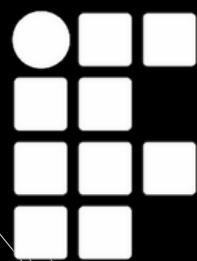
Campus  
Tianguá

# **Padrões de Projeto:**

# **DECORATOR E MEMENTO**

**Equipe:**  
**Felipe Freire ,**  
**Francisco Wanderson,**  
**Lucas Portela,**  
**Lucas Souza,**  
**Natã dos Anjos**





**INSTITUTO  
FEDERAL**

Ceará

---

Campus  
Tianguá

# **INTRODUÇÃO GERAL E PADRÕES DE PROJETO**

# O QUE SÃO PADRÕES DE PROJETO ?

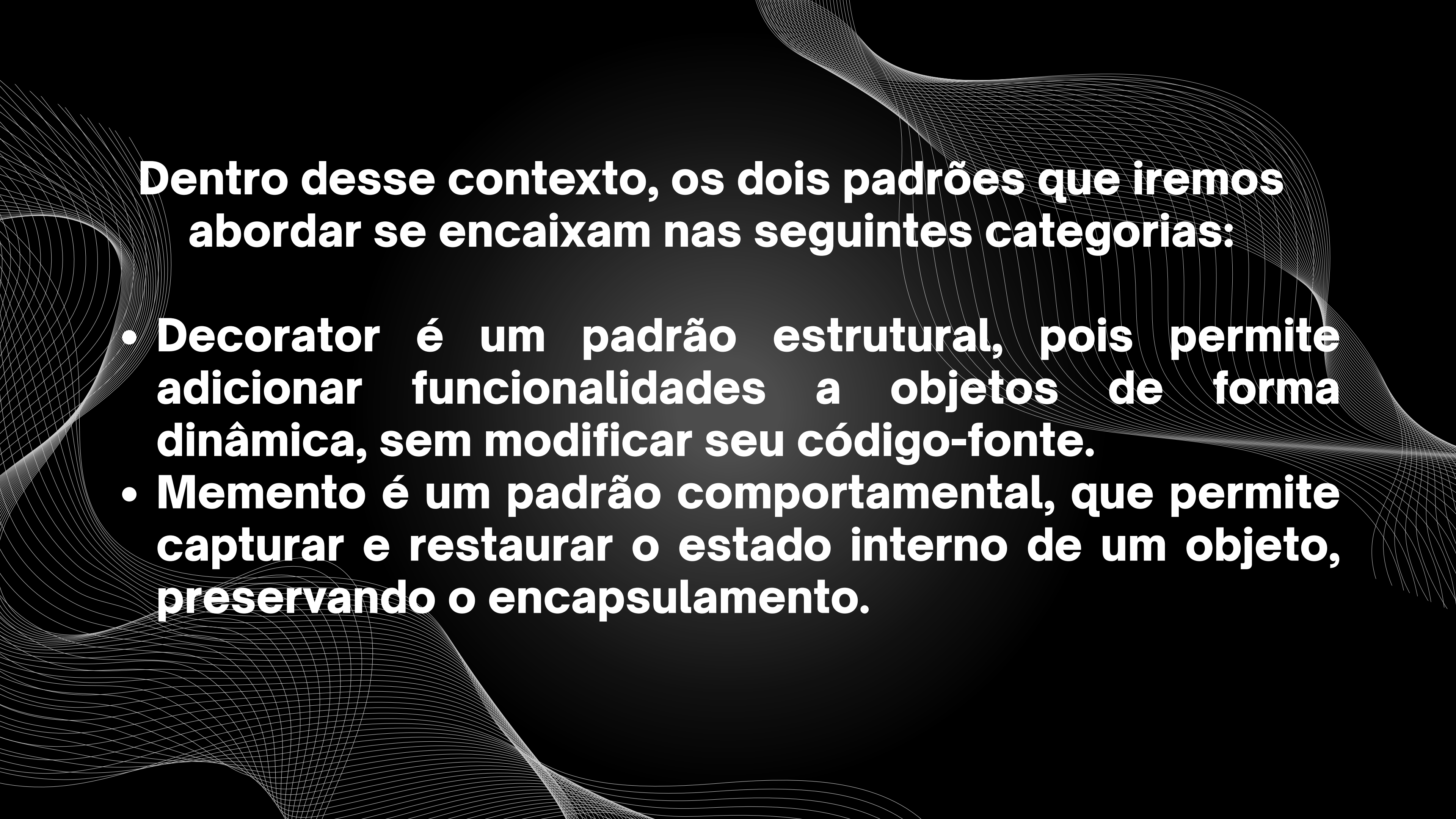
**Padrões de projeto são soluções reutilizáveis para problemas recorrentes que surgem durante o desenvolvimento de software orientado a objetos. Eles não são pedaços prontos de código, mas sim estratégias e boas práticas que ajudam a escrever código mais limpo, flexível e de fácil manutenção.**



# CATEGORIAS DOS PADRÕES DE PROJETOS

**Esses padrões são geralmente classificados em três grandes categorias:**

- Padrões Criacionais: lidam com a criação de objetos.**
- Padrões Estruturais: tratam da composição de classes e objetos.**
- Padrões Comportamentais: focam na comunicação entre objetos e como eles interagem.**

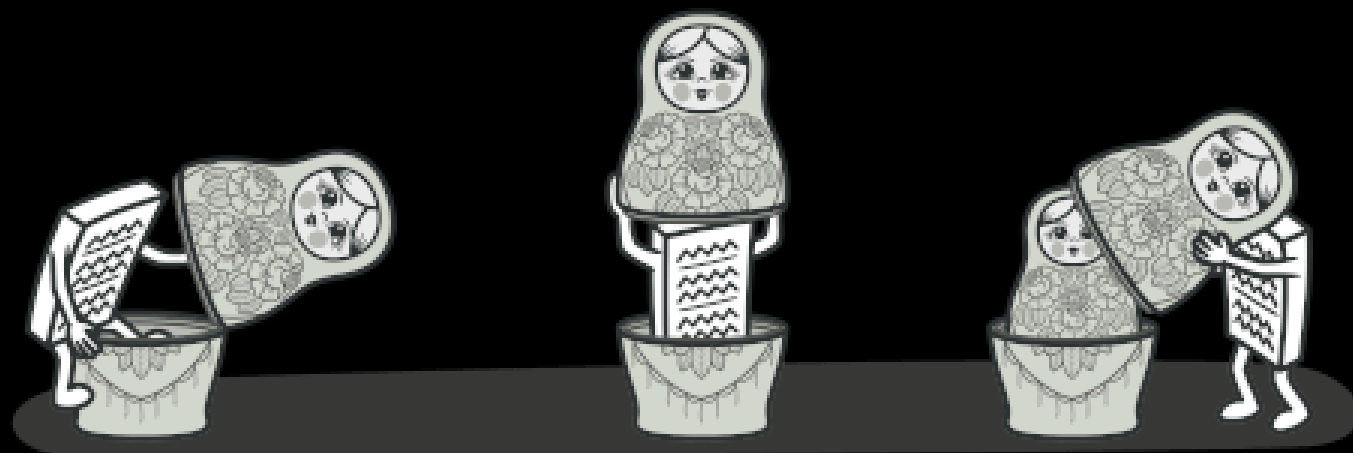
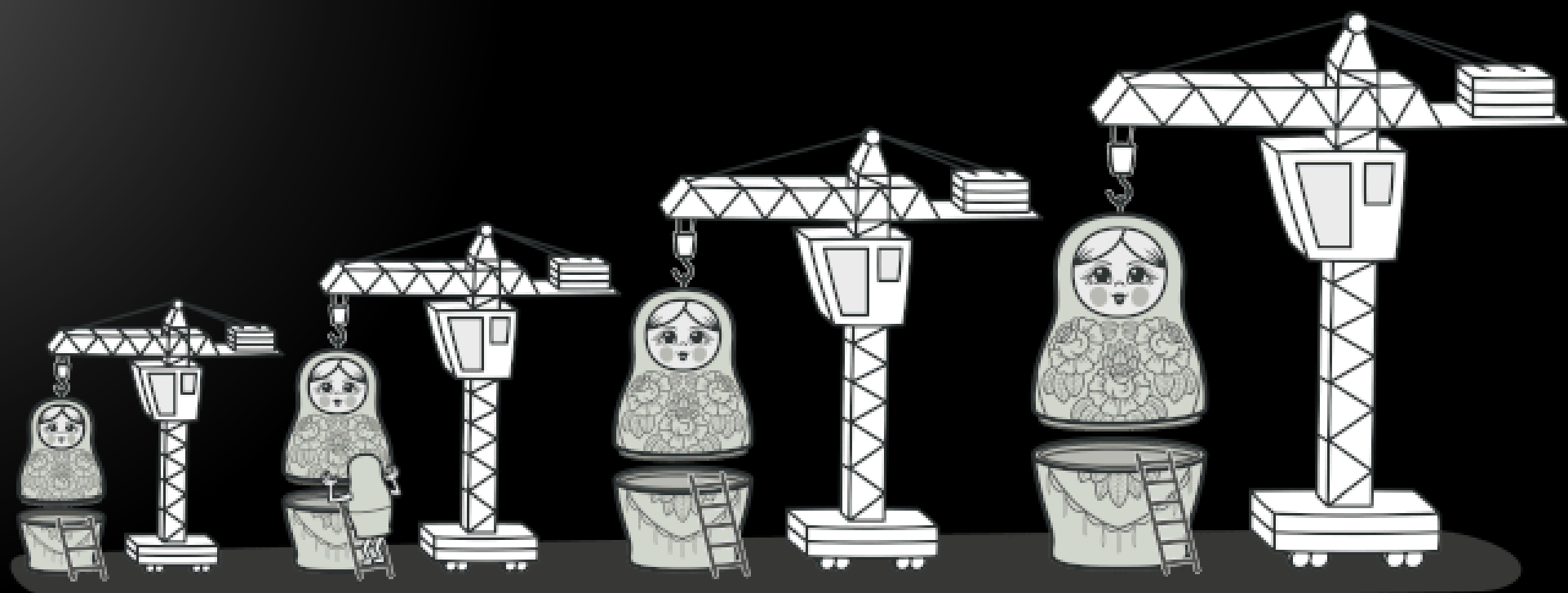


**Dentro desse contexto, os dois padrões que iremos abordar se encaixam nas seguintes categorias:**

- **Decorator é um padrão estrutural, pois permite adicionar funcionalidades a objetos de forma dinâmica, sem modificar seu código-fonte.**
- **Memento é um padrão comportamental, que permite capturar e restaurar o estado interno de um objeto, preservando o encapsulamento.**

# DECORATOR

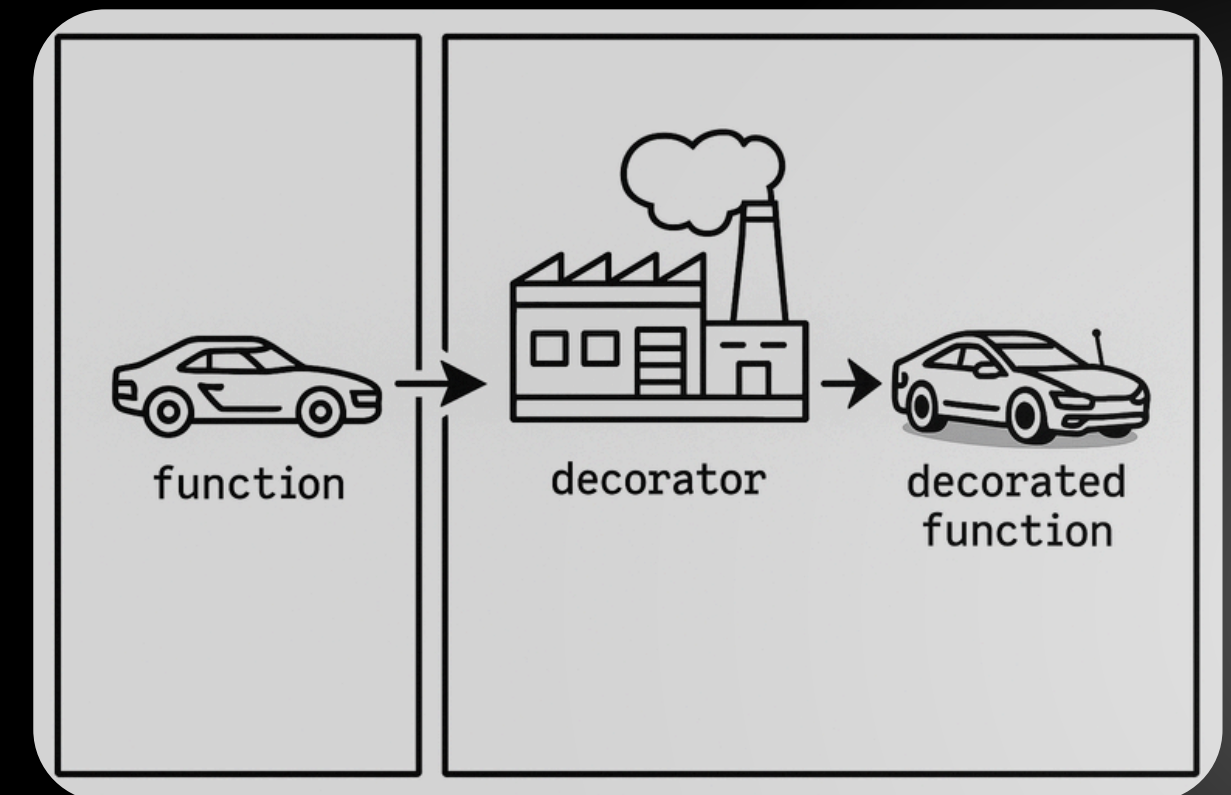
---



# INTRODUÇÃO AO PADRÃO DECORATOR

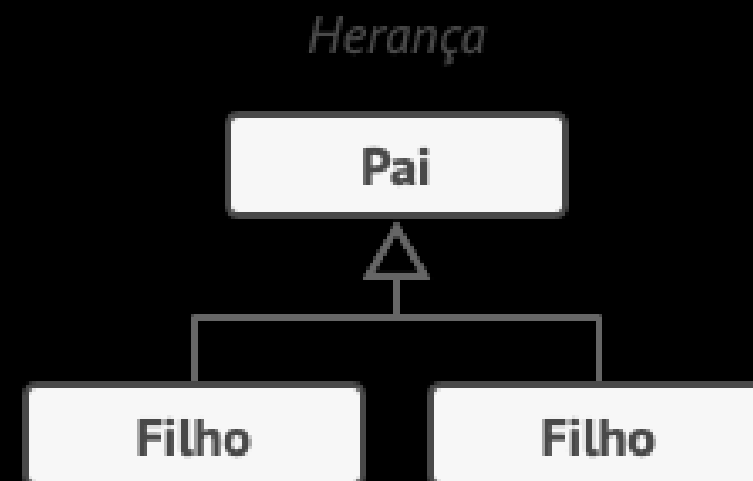
- O *Decorator* é um padrão de projeto estrutural que permite adicionar novos comportamentos a um objeto de forma dinâmica, sem alterar sua estrutura original.

## FUNCIONAMENTO BÁSICO:



# OBJETIVO DO DECORATOR

- *Adicionar funcionalidades a um objeto de forma dinâmica.*
- *Evitar muitas subclasses e herança rígida.*
- *Permitir extensão de comportamento sem alterar o código original.*





# VANTAGENS DE USAR O DECORATOR

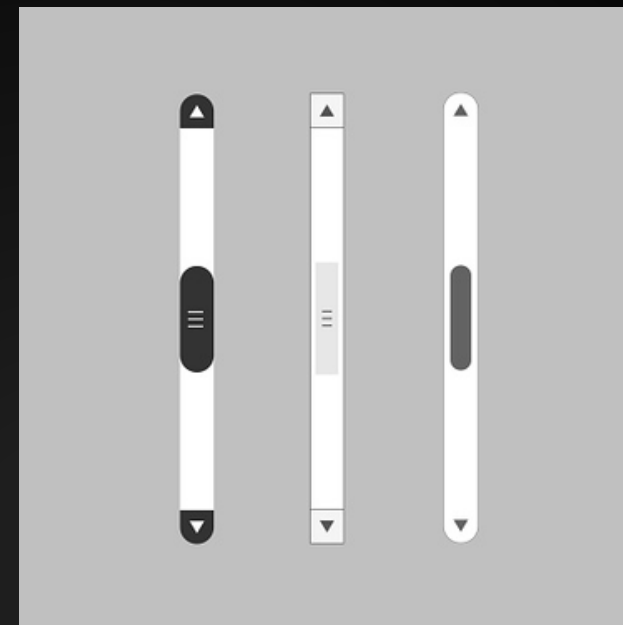
- É mais flexível que a herança, pois adiciona responsabilidade em tempo de execução e não em tempo de compilação
- Podemos ter qualquer número de decoradores e em qualquer ordem
- Estende a funcionalidade do objeto sem afetar outros objetos

## ANALOGIA COM O MUNDO REAL



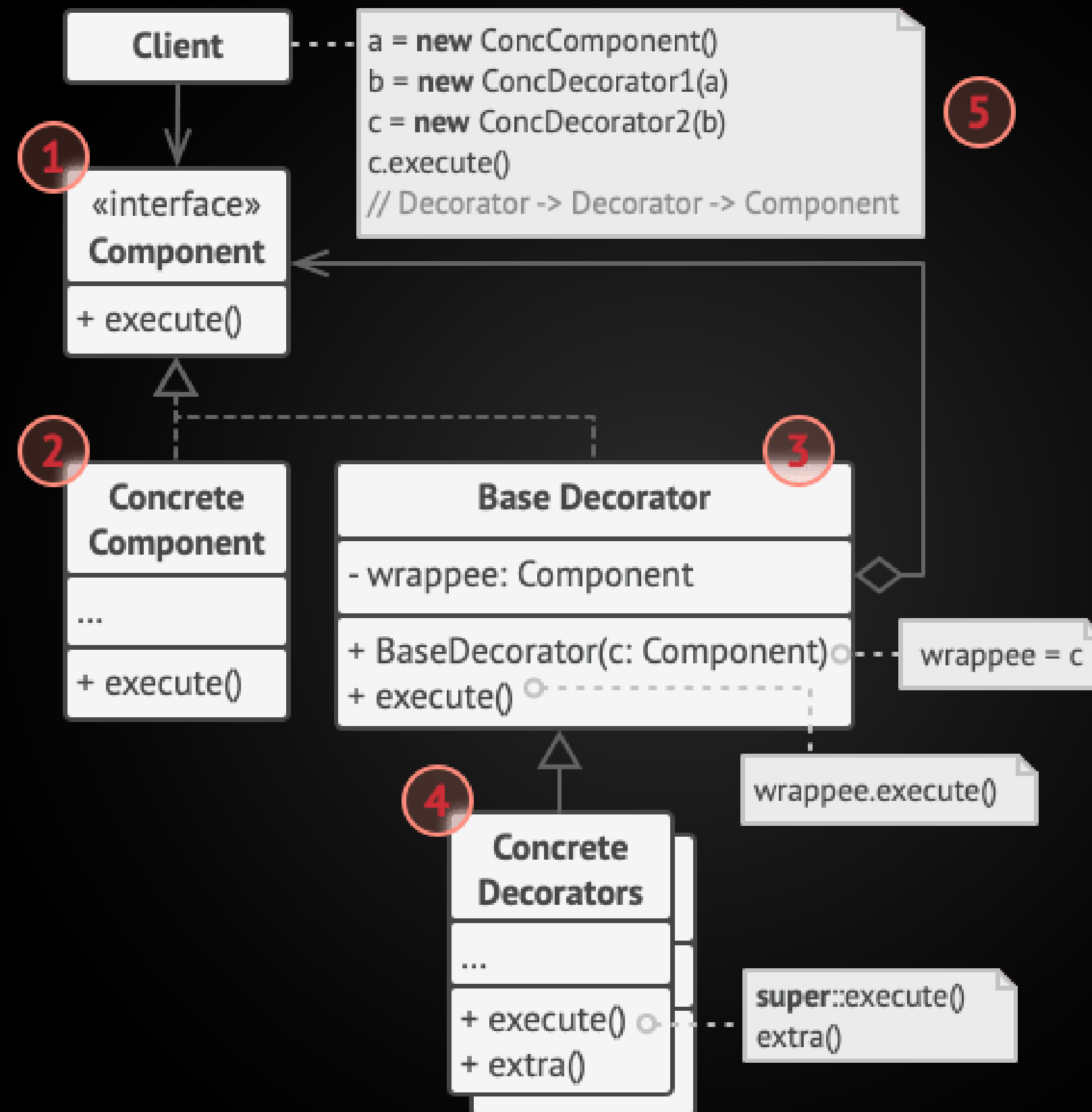
# APLICAÇÕES REAIS

- ◆ Interface Gráfica (GUI)
- ◆ Sistema de Logs
- ◆ Filtros em Servidores Web
- ◆ Editor de Texto
- ◆ Bibliotecas de Notificação



**negrito**  
*itálico*

# ESTRUTURA DO PADRÃO DECORATOR





# SITUAÇÃO 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.
- Os coquetéis são feitos da combinação de uma bebida base e vários outros adicionais que compõe a bebida.



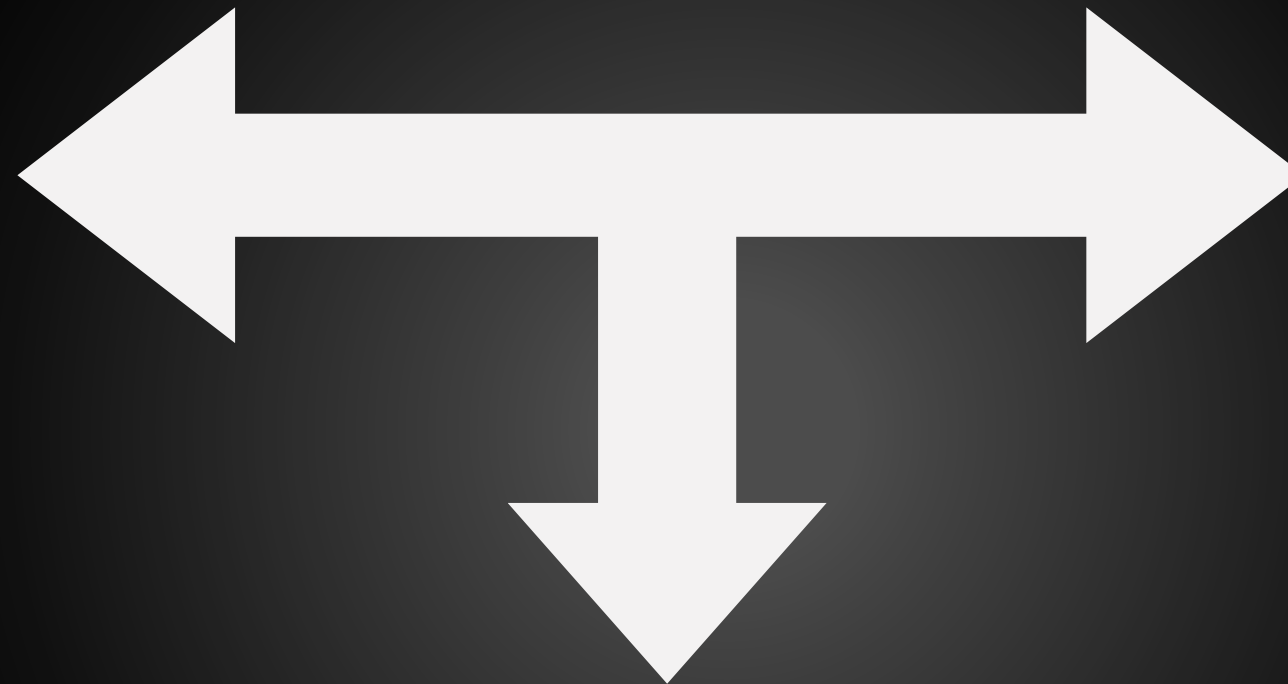
# SITUAÇÃO PROBLEMA

## BEBIDAS

- CACHAÇA
- RUM
- VODKA
- TEQUILA

## ADICIONAIS

- LIMÃO
- REFRIGERANTE
- SUCO
- LEITE CONDENSADO
- GELO
- AÇÚCAR



## COQUETÉIS

- VODKA + SUCO + GELO + AÇÚCAR
- TEQUILA + LIMÃO + SAL
- CACHAÇA + LEITE CONDENSADO + AÇÚCAR + GELO

# COMO RESOLVER ESSE PROBLEMA ?

- Poderíamos utilizar como uma solução simples uma classe abstrata Coquetel extremamente genérica e, para cada tipo de coquetel construir uma classe concreta.

```
1  public abstract class Coquetel {  
2      String nome;  
3      double preco;  
4  
5      public String getNome() {  
6          return nome;  
7      }  
8  
9      public double getPreco() {  
10         return preco;  
11     }  
12 }
```

FONTE: <https://brizeno.wordpress.com/2011/08/31/decorator/>

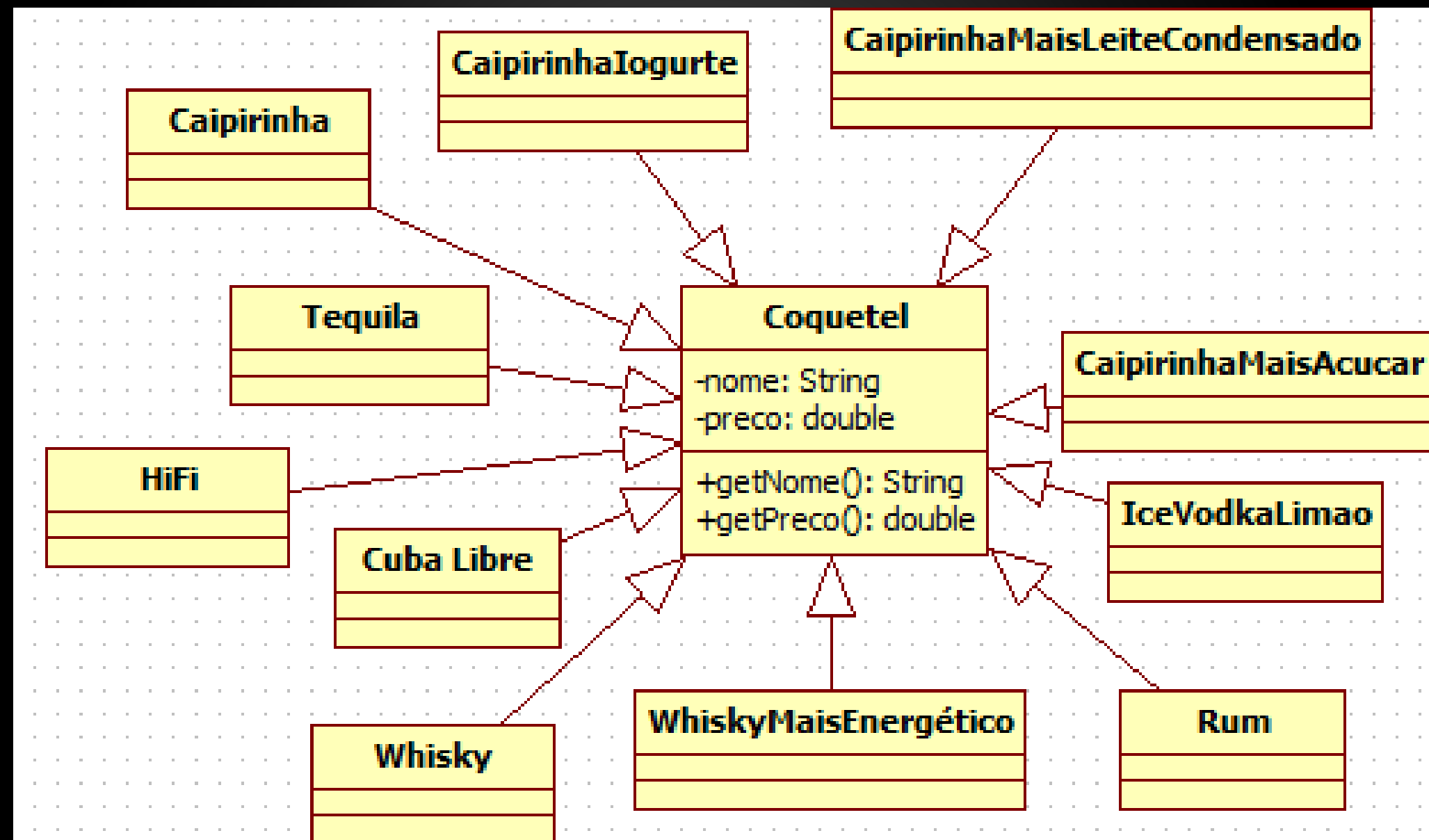
```
public class Caipirinha extends Coquetel {  
    public Caipirinha() {  
        nome = "Caipirinha";  
        preco = 3.5;  
    }  
}
```

FONTE: <https://brizeno.wordpress.com/2011/08/31/decorator/>



# COMO RESOLVER ESSE PROBLEMA ?

- Desse modo, teríamos um diagrama desse modo:

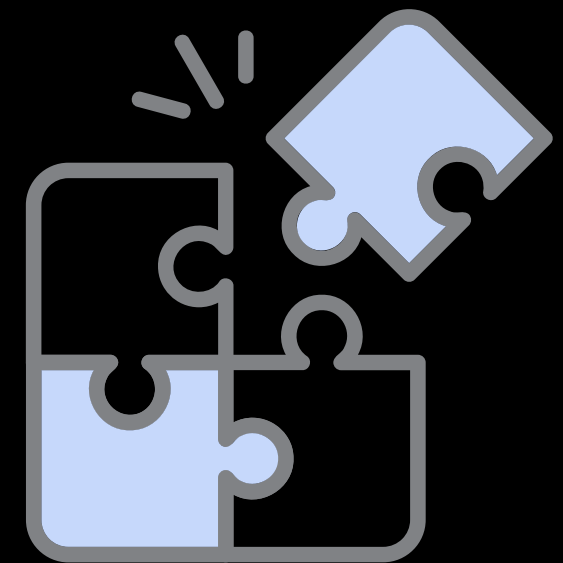


# COMO RESOLVER ESSE PROBLEMA ?

- O cliente pode desejar adicionar doses extras de determinados tipos de adicionais
- Desse modo, seria muito complicado modelar o sistema para prever todas as possibilidades!
- Então, como resolver o problema ?

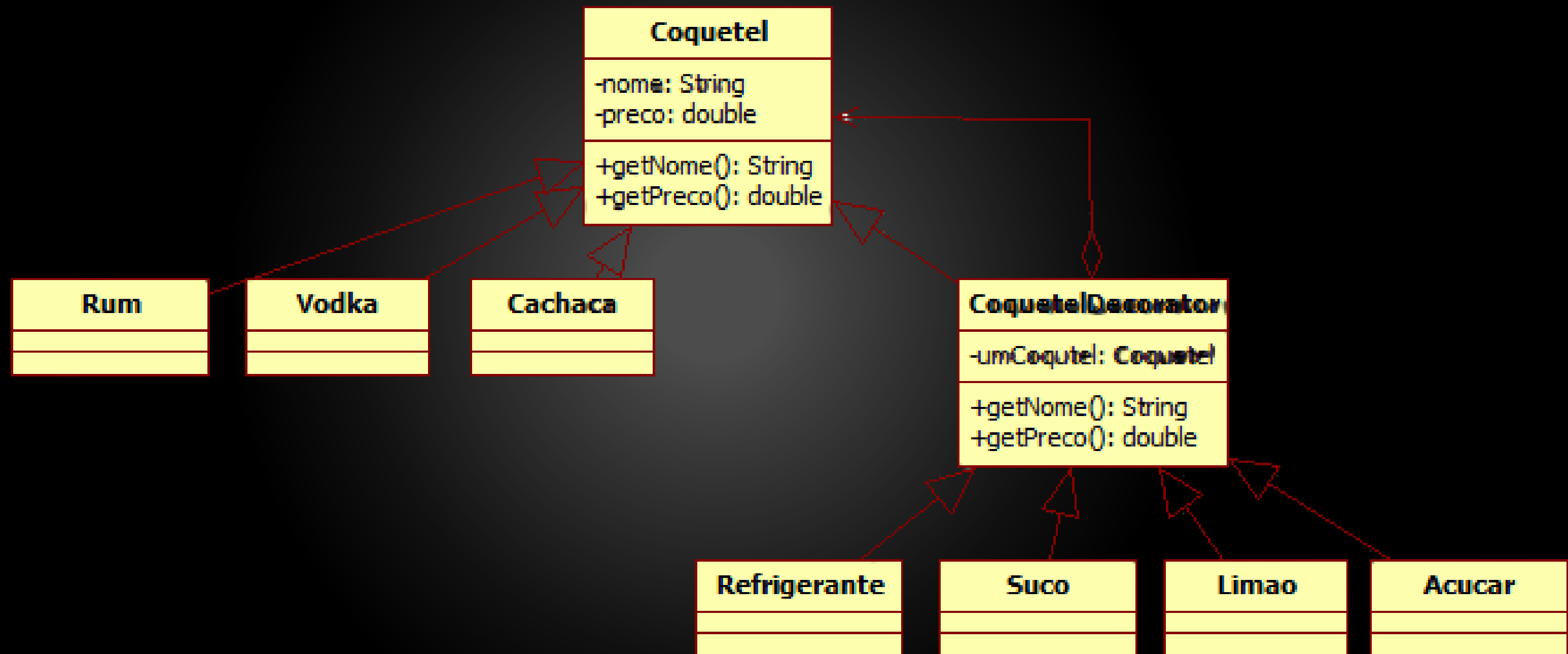
# SOLUÇÃO

- Queremos que, dado um objeto Coquetel, seja possível adicionar funcionalidades a ele, e somente a ele.
- “Dinamicamente, agregar responsabilidades adicionais a objetos. Os Decorators fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades.” [1]





# SOLUÇÃO



FONTE: <https://brizenowordpress.com/2011/08/31/decorator/>

# SOLUÇÃO

- Todos os objetos possuem o mesmo tipo Coquetel, esta classe define o que todos os objeto possuem sendo igual a classe já feita antes.
- As classes de bebidas concretas definem apenas os dados relativos a ela.
- Todas as classes de bebidas possuirão a mesma estrutura.
- A classe Decorator abstrata define que todos os decoradores possuem um objeto Coquetel, ao qual decoram, e um método que é aplicado a este objeto.
- Como o decorador também é um Coquetel ele herda os atributos nome e preço.
- Nas classes concretas apenas definimos os modificadores que serão aplicados, de maneira semelhante as classes de bebidas concretas

# CÓDIGO EM JAVA

```
Coquetel.java X
1 package exemplo1;
2
3 public abstract class Coquetel {
4     String nome;
5     double preco;
6
7     public String getNome() {
8         return nome;
9     }
10
11     public double getPreco() {
12         return preco;
13     }
14 }
```



# CÓDIGO EM JAVA

```
package exemplo1;

public abstract class CoquetelDecorator extends Coquetel {
    Coquetel coquetel;

    public CoquetelDecorator(Coquetel umCoquetel) {
        coquetel = umCoquetel;
    }

    @Override
    public String getNome() {
        return coquetel.getNome() + " + " + nome;
    }

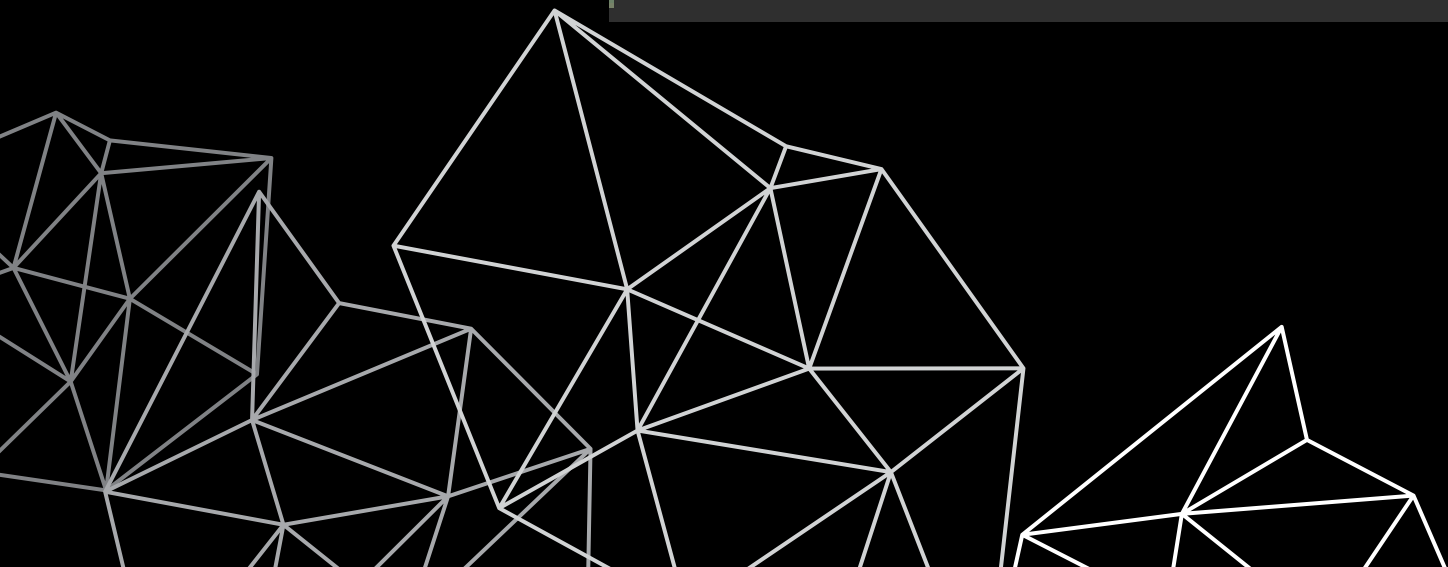
    public double getPreco() {
        return coquetel.getPreco() + preco;
    }
}
```

# CÓDIGO EM JAVA

```
package exemplo1;

public class Refrigerante extends CoquetelDecorator {

    public Refrigerante(Coquetel umCoquetel) {
        super(umCoquetel);
        nome = "Refrigerante";
        preco = 1.0;
    }
}
```

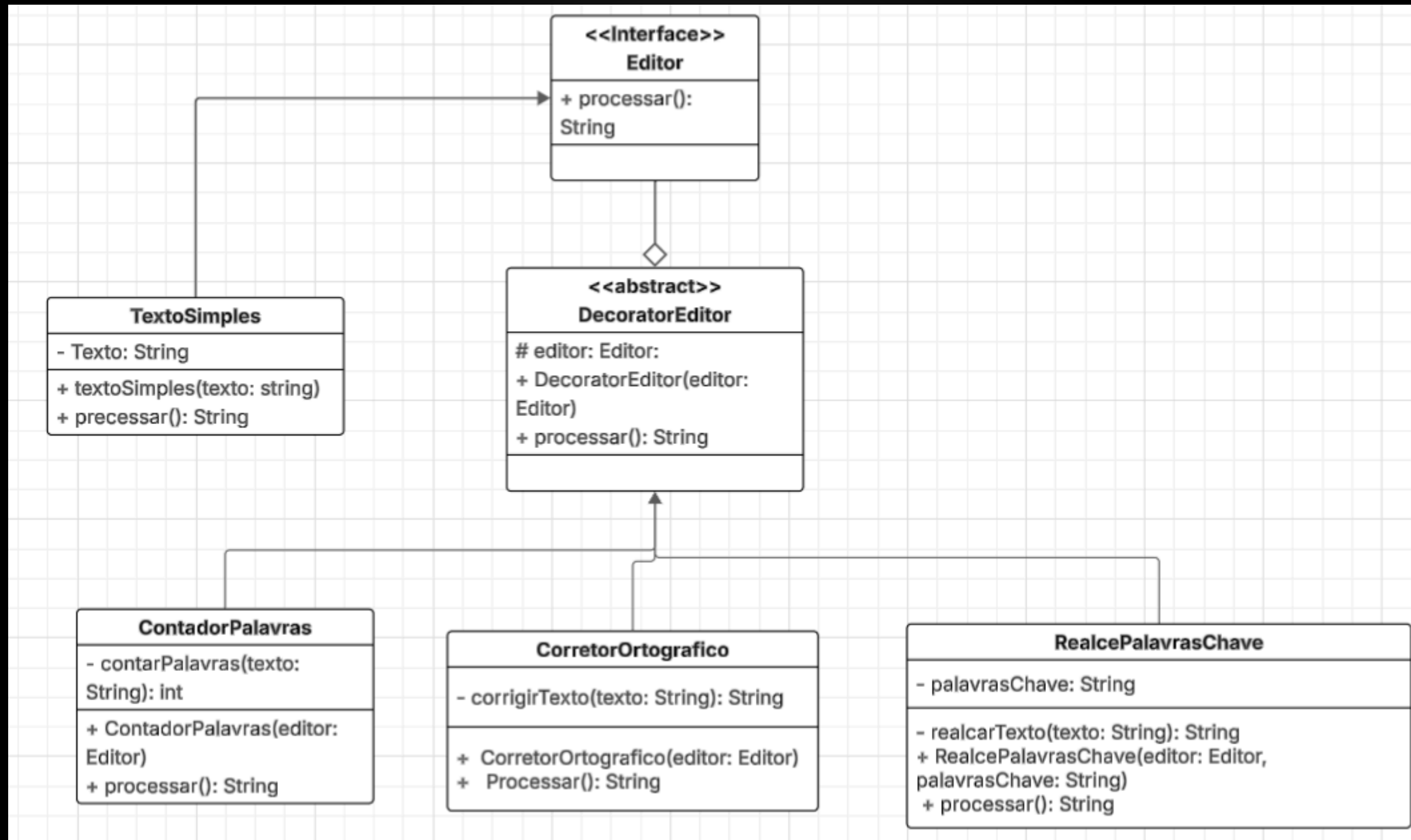


# SITUAÇÃO PROBLEMA 2

- Exemplo de um editor de texto que possua diversas funcionalidades.
- O usuário gostaria de utilizar algumas das diversas funcionalidades como correção ortográfica, contagem de palavras etc.
- O usuário teria a opção de utilizar mais de uma funcionalidade ou nenhuma.



# SOLUÇÃO



FONTE: Próprio autor

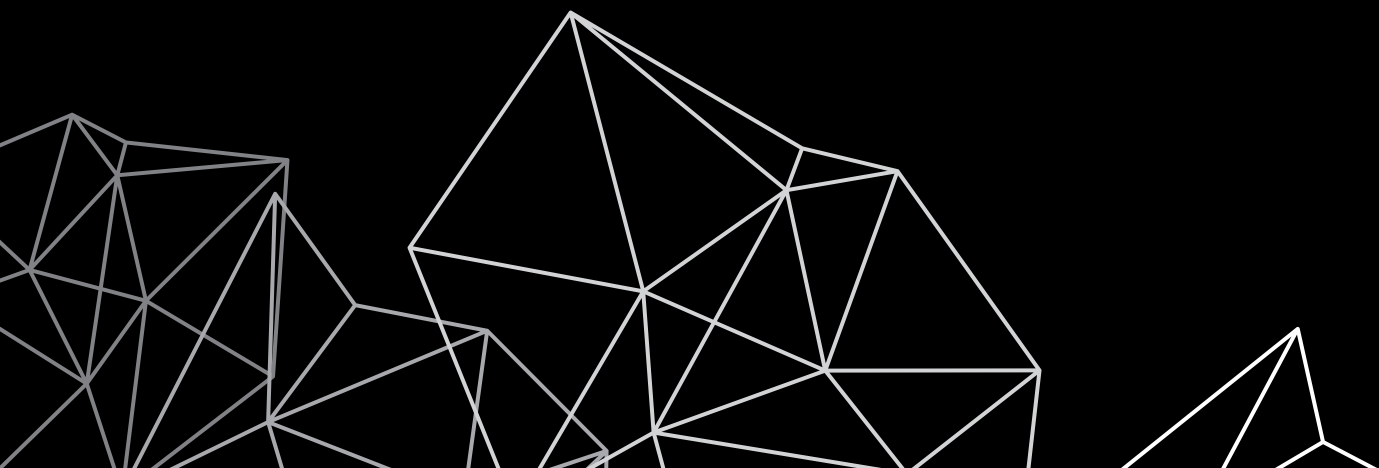
# CÓDIGO EM JAVA

```
package exemplo2;

//Decorador abstrato
public abstract class DecoradorEditor implements Editor {
    protected Editor editor;

    public DecoradorEditor(Editor editor) {
        this.editor = editor;
    }

    public String processar() {
        return editor.processar();
    }
}
```





# CÓDIGO EM JAVA

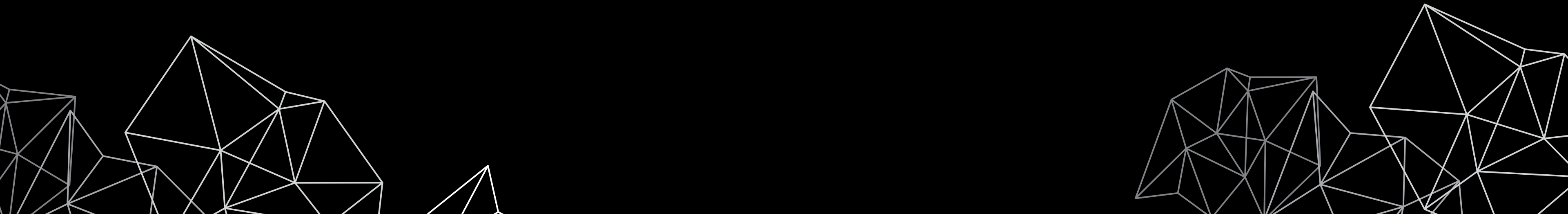
```
//Decoradores concretos
public class CorretorOrtografico extends DecoradorEditor {

    public CorretorOrtografico(Editor editor) {
        super(editor);
    }

    @Override
    public String processar() {
        String textoProcessado = editor.processar();
        String textoCorrigido = corrigirTexto(textoProcessado);
        return textoCorrigido + "\n[CorretorOrtografico] Verificação ortográfica aplicada.";
    }

    private String corrigirTexto(String texto) {
        // Lógica de correção ortográfica simplificada (apenas para exemplo)
        return texto.replace("txto", "texto")
            .replace("ortografico", "ortográfico")
            .replace("simpres", "simples");
    }
}
```

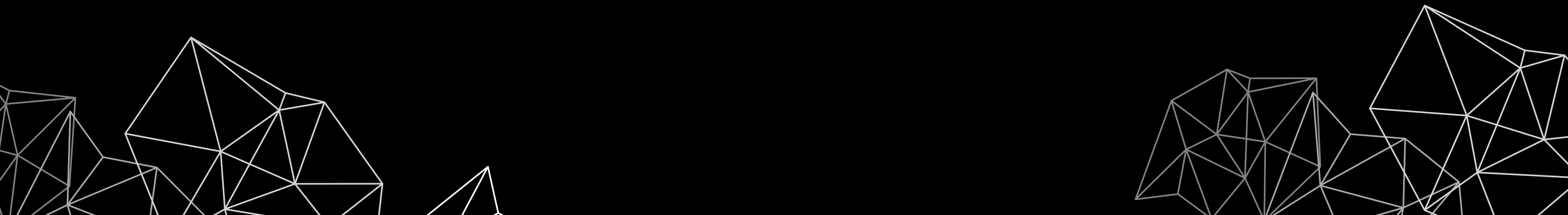
**COMO ESSE PADRÃO PODERIA SER UTILIZADO NO SISTEMA  
DESENVOLVIDO PELA EQUIPE ?**

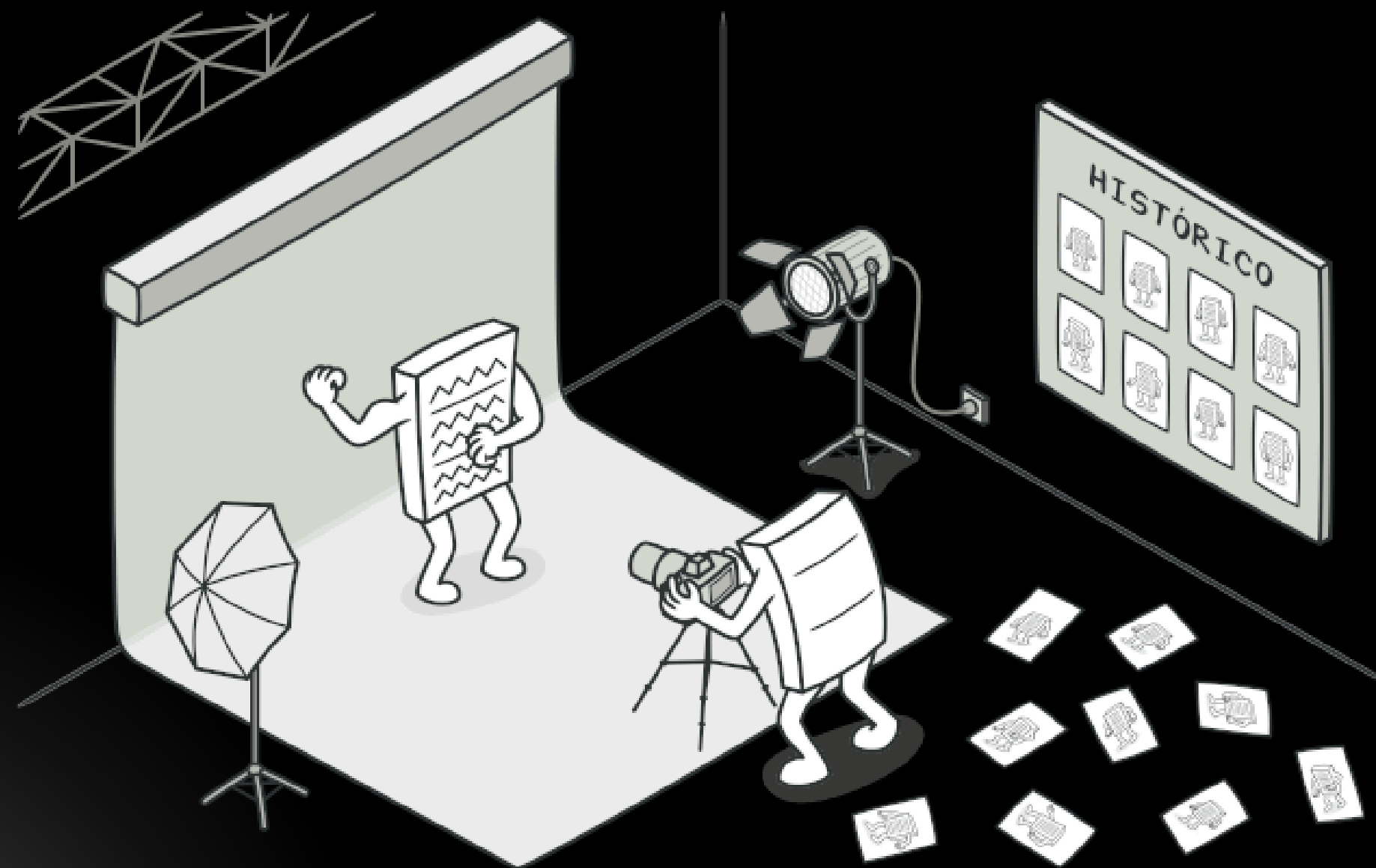


A academia oferece um plano base, e o aluno pode adicionar serviços opcionais como:

- Aulas individuais com personal
- Acompanhamento nutricional
- Aulas de Pilates

Cada serviço adicional aumenta o custo do plano, e o sistema deseja permitir que os alunos escolham essas opções de forma mais flexível.





# MEMENTO

---

# INTRODUÇÃO AO PADRÃO MEMENTO

- O Memento é um padrão de projeto comportamental que permite que você salve e restaure o estado anterior de um objeto sem revelar os detalhes de sua implementação.
- É como se o objeto tirasse uma "foto" de si mesmo. Mais tarde, ele pode usar essa foto para voltar exatamente ao ponto em que estava



Exemplo: funcionalidade "desfazer" em editores



# VANTAGENS DO MEMENTO

## **Preserva o Encapsulamento**

- O objeto salva seu próprio estado sem mostrar seus dados internos.
- Só o próprio objeto consegue ver ou usar esse estado salvo.

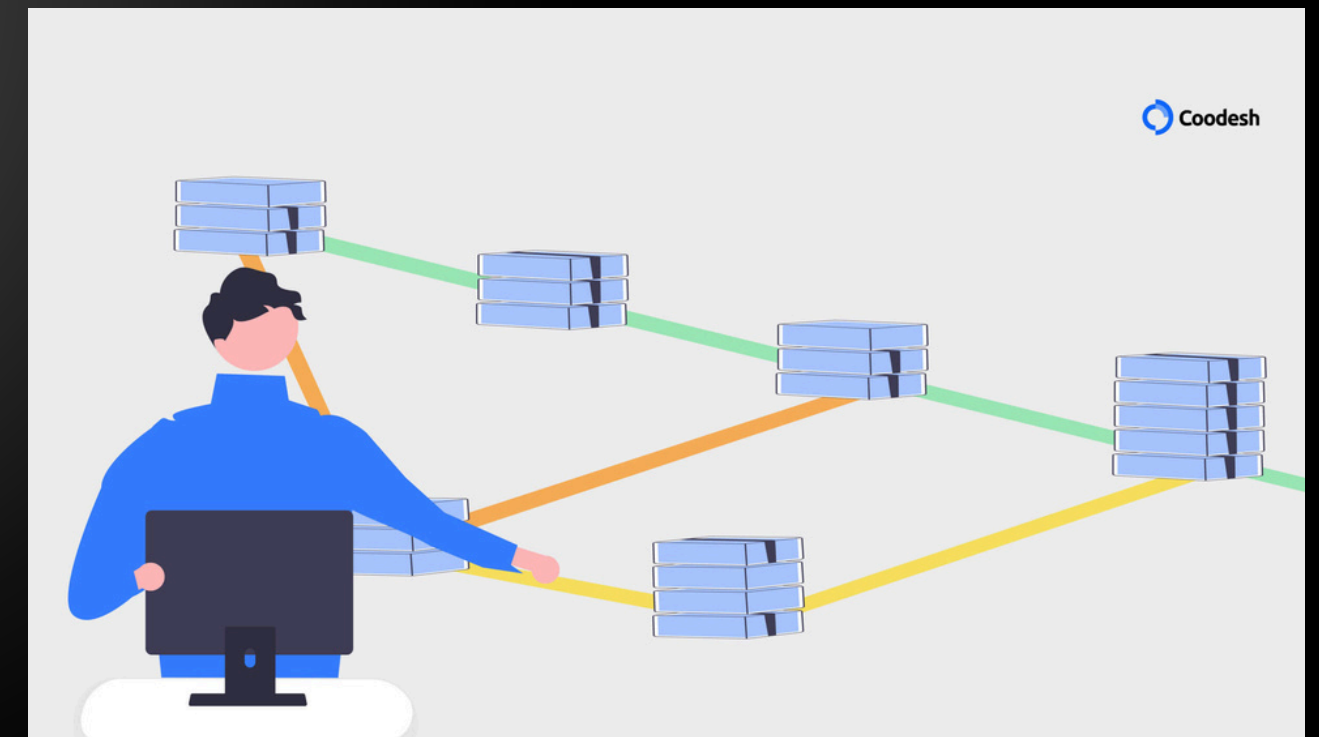
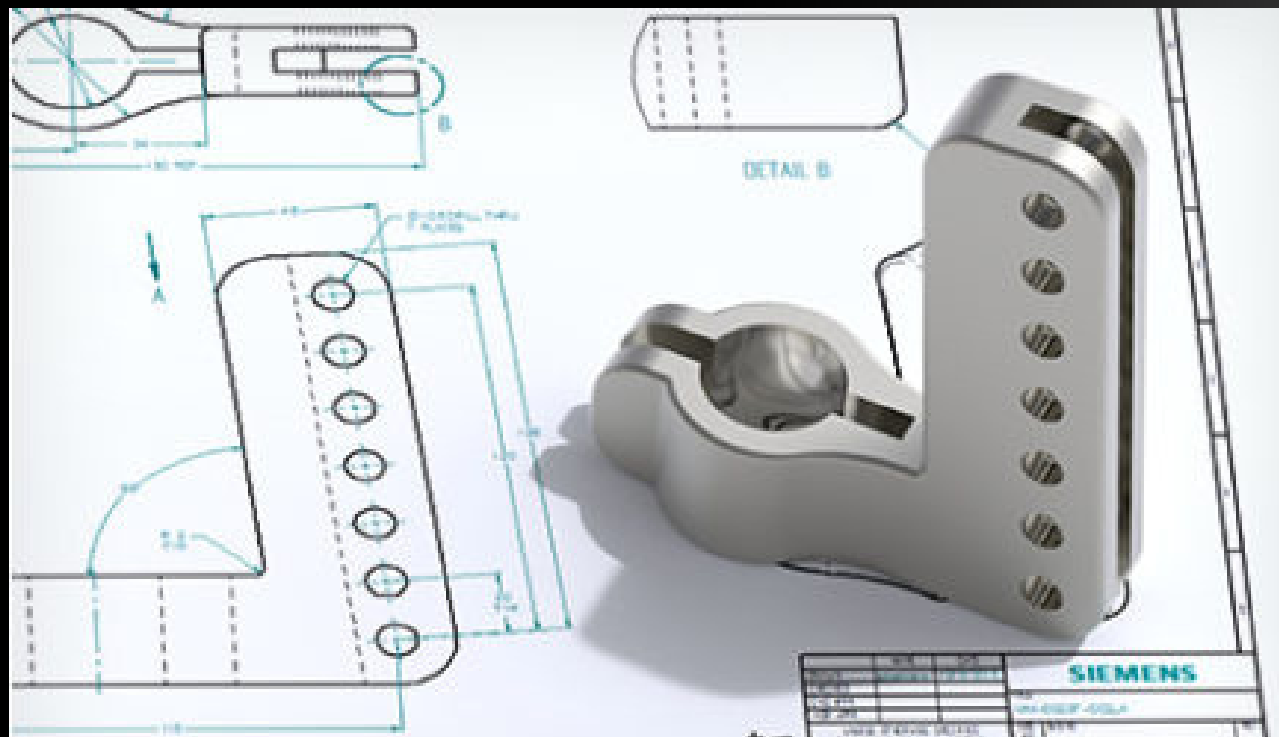
## **Código Limpo e Organizado**

- O próprio objeto sabe como salvar e voltar ao seu estado anterior.
- Quem guarda os estados (como um histórico) só armazena, sem saber o que tem dentro.



# APLICAÇÕES REAIS

- Editores de texto (desfazer/refazer)
- Jogos (save/load)
- Sistemas de desenho e design (ex: CAD)
- Controle de versão e fluxos de formulários



# PROBLEMA

## Cenário

- Editor de texto com funcionalidades: edição, formatação, imagens, etc.
- Requisito: permitir desfazer operações anteriores.

## Problemas da abordagem direta

- Precisa copiar todo o estado interno do objeto.
- Expor campos privados quebra o encapsulamento.
- Toda mudança interna exige alterações no código do histórico.

## O dilema

- Expor o estado → Sistema frágil e acoplado.
- Esconder o estado → Não é possível copiar.



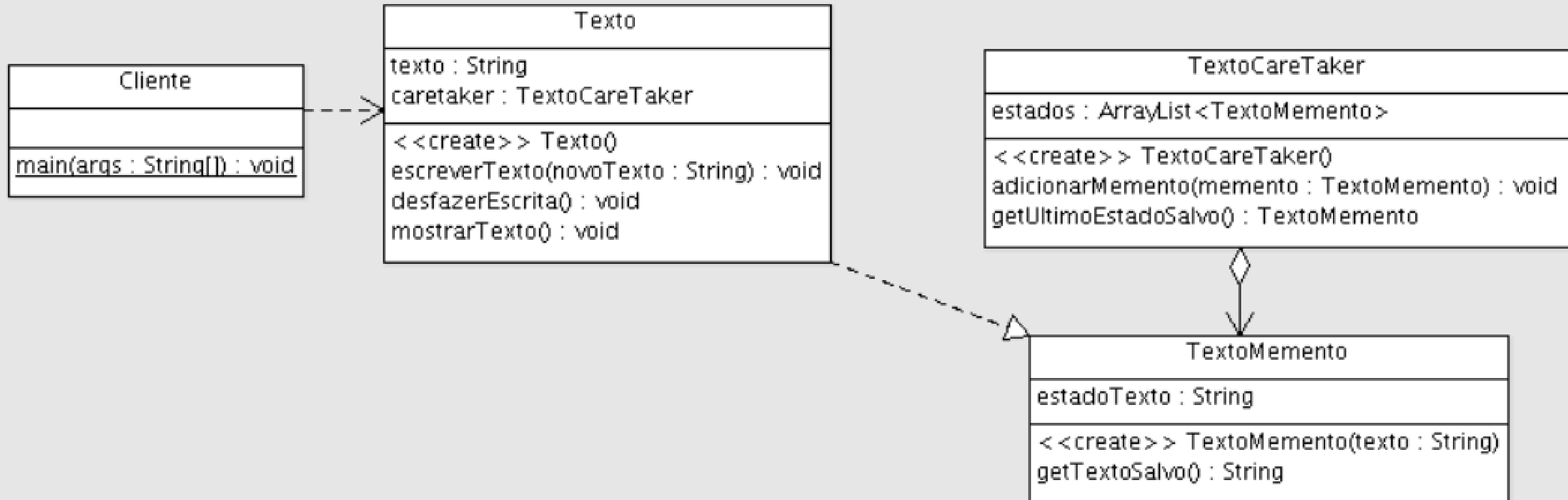
# SOLUÇÃO

- Permitir capturar ou restaurar o estado sem quebrar encapsulamento.
- O próprio objeto cria sua "foto" (memento) e o entrega ao histórico.
- A restauração ocorre de forma segura e desacoplada



# DIAGRAMA UML

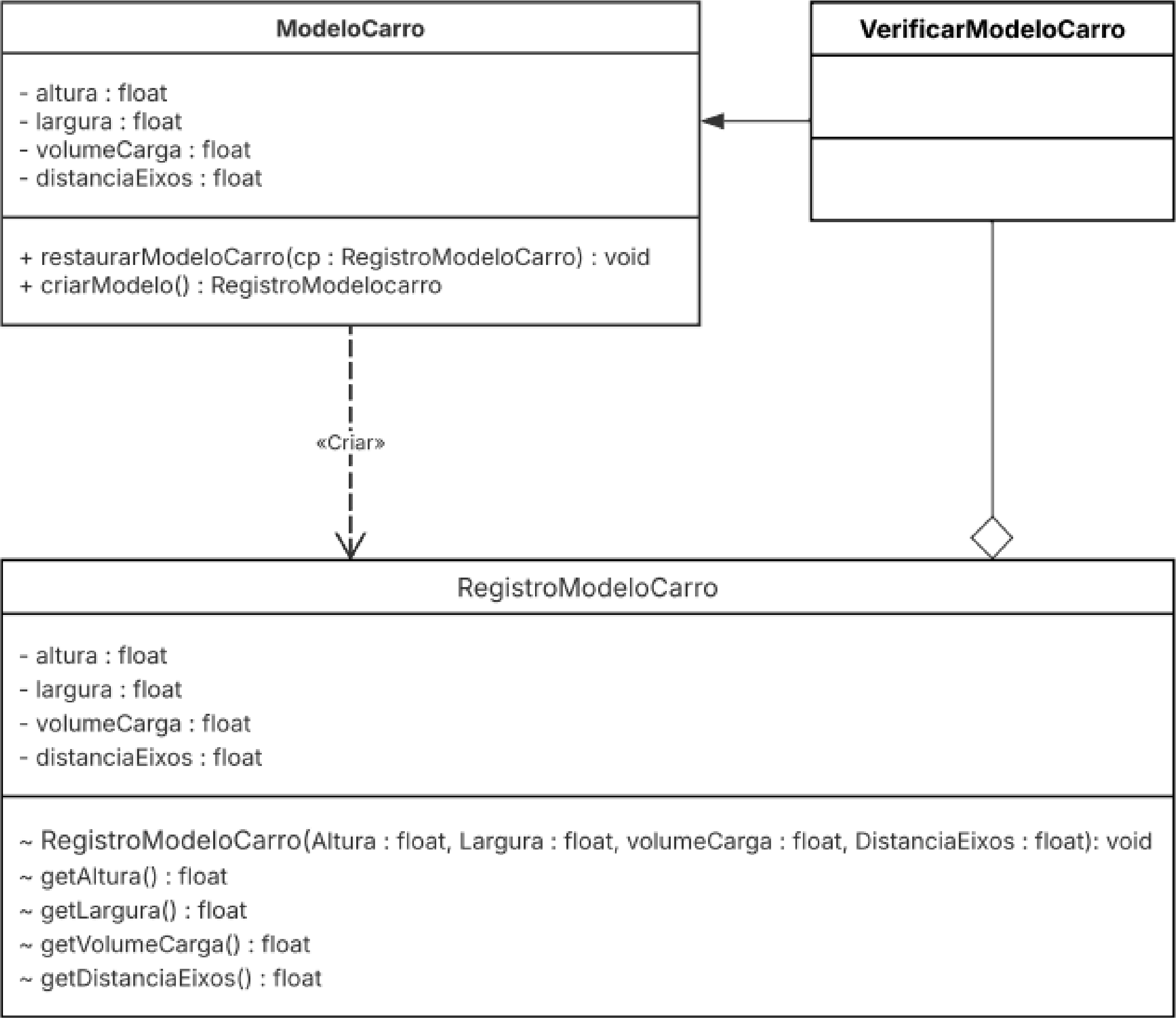
- Projeto Texto memento





# DIAGRAMA UML

- Projeto Carro memento



# **COMO O PADRÃO MEMENTO PODE SER APLICADO NO TRABALHO DA N1**

## **Ficha de Treino Personalizada**

- **Um aluno tem uma ficha com exercícios, séries e cargas.**
- **Ao alterar a ficha (ex: trocar exercícios ou pesos), o sistema salva o estado anterior**
- **Se o instrutor quiser "desfazer" uma mudança, ele pode restaurar a versão anterior.**


# REFERÊNCIAS



## Padrão Decorator

01. <https://brizeno.wordpress.com/2011/08/31/decorator/>
02. <https://refactoring.guru/pt-br/design-patterns/decorator>
03. <http://www.labies.uff.br/padroesdr/ideas/showPattern/decorator>

## Padrão Memento

01. <https://refactoring.guru/pt-br/design-patterns/memento>
  02. <http://www.labies.uff.br/padroesdr/questions/5/patterns/memento>
  03. <https://brizeno.wordpress.com/category/padroes-de-projeto/memento/>
- 

The background is a dark gradient with white, wavy, wireframe-like lines that create a sense of depth and movement, framing the central text.

**THANK YOU**