DECORATOR

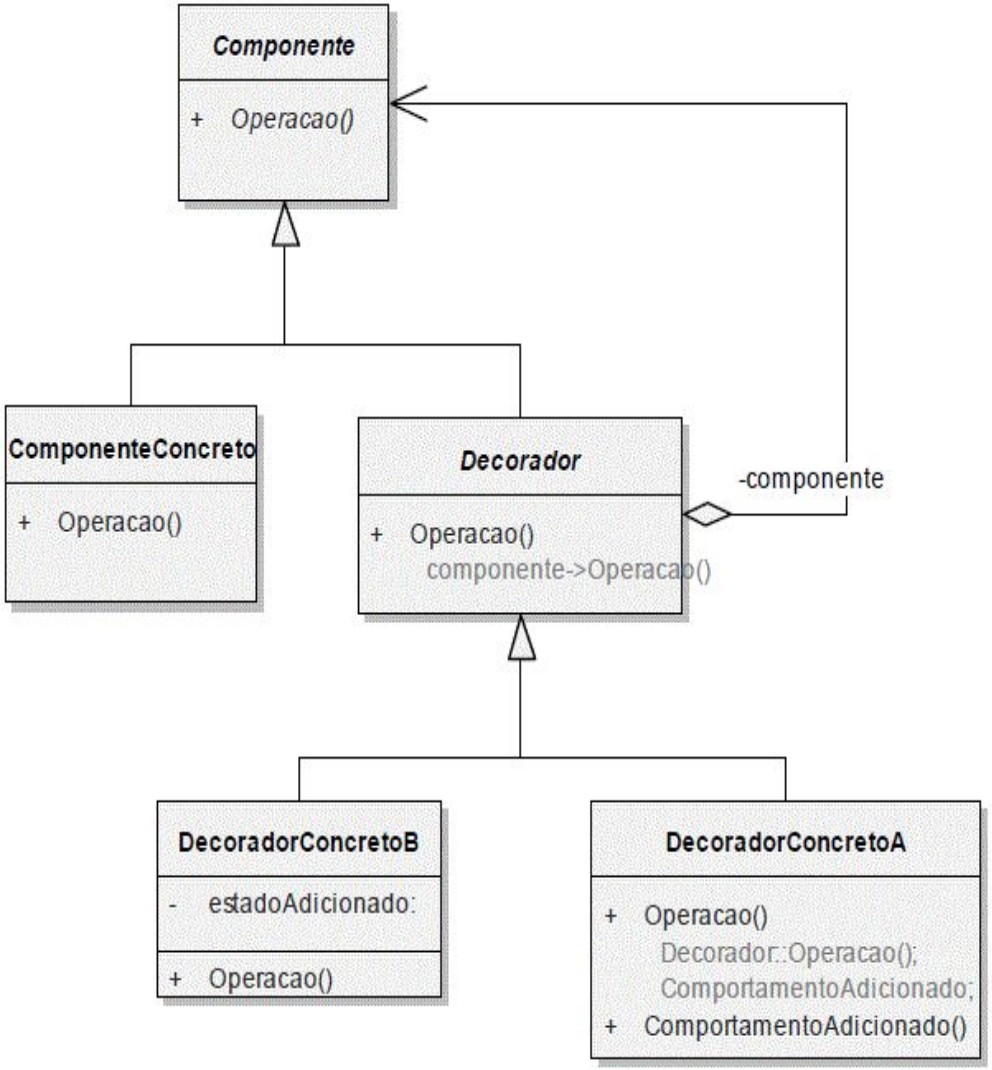
Um design pattern por semana

github.com/bennersaude/UmDesignPatternPorSemana

“

Dinamicamente, agregar responsabilidades adicionais a objetos. Os Decorators fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades.

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.



Classes/Objetos participantes do padrão:

1. **Componente** - Define a interface para objetos que podem ter responsabilidades adicionadas a eles dinamicamente;
2. **ComponenteConcreto** - Define um objeto para o qual responsabilidades adicionais podem ser anexadas;
3. **Decorador** - Mantém uma referência para um objeto Componente e define uma interface compatível com interface de componente;
4. **DecoradorConcreto** - Adiciona responsabilidades ao componente;

PROBLEMA

Sistema para bar com vários tipos de coquetéis:

Conjunto de bebidas:

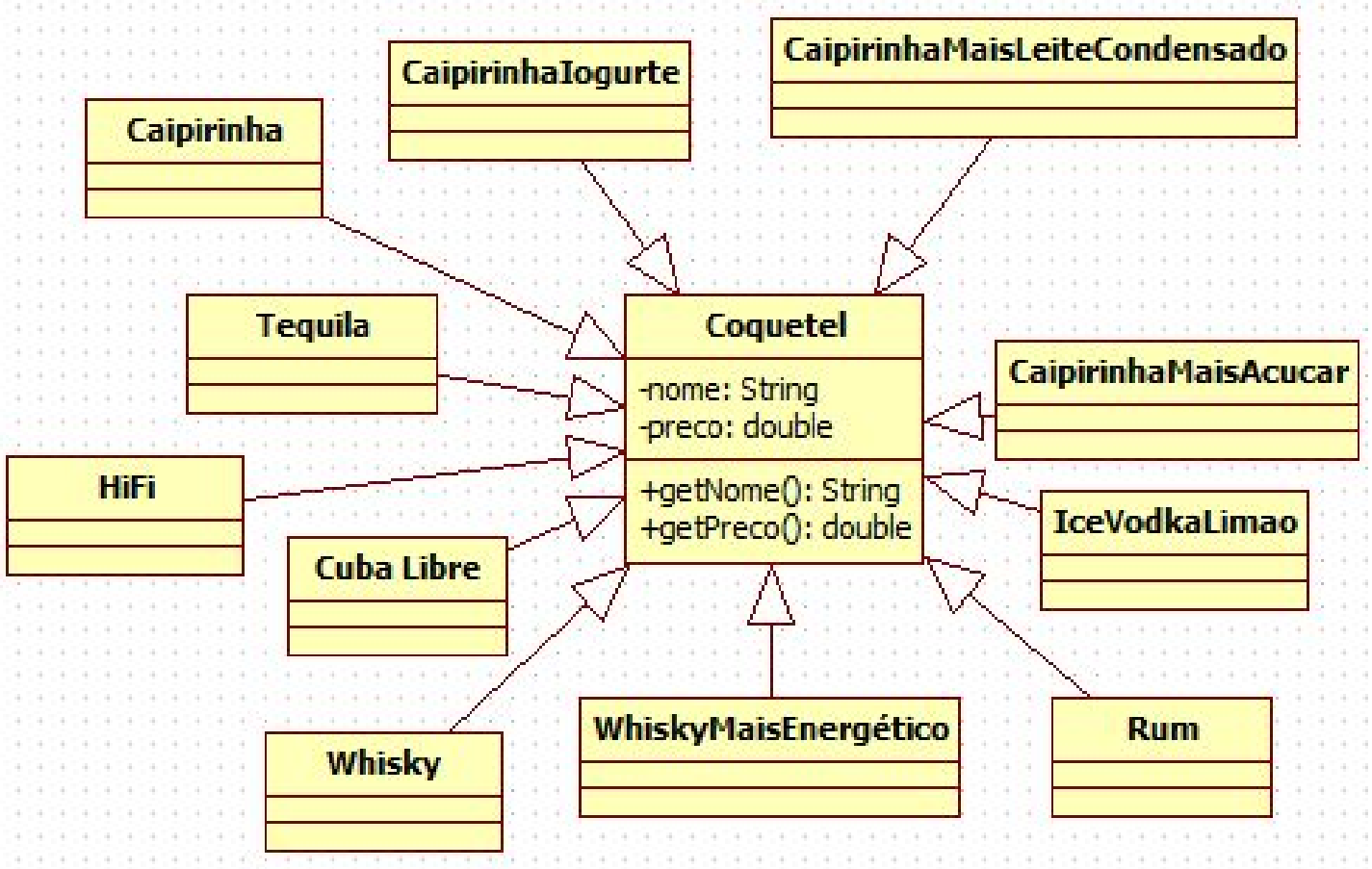
- Cachaça
- Rum
- Vodka
- Tequila

Conjunto de adicionais:

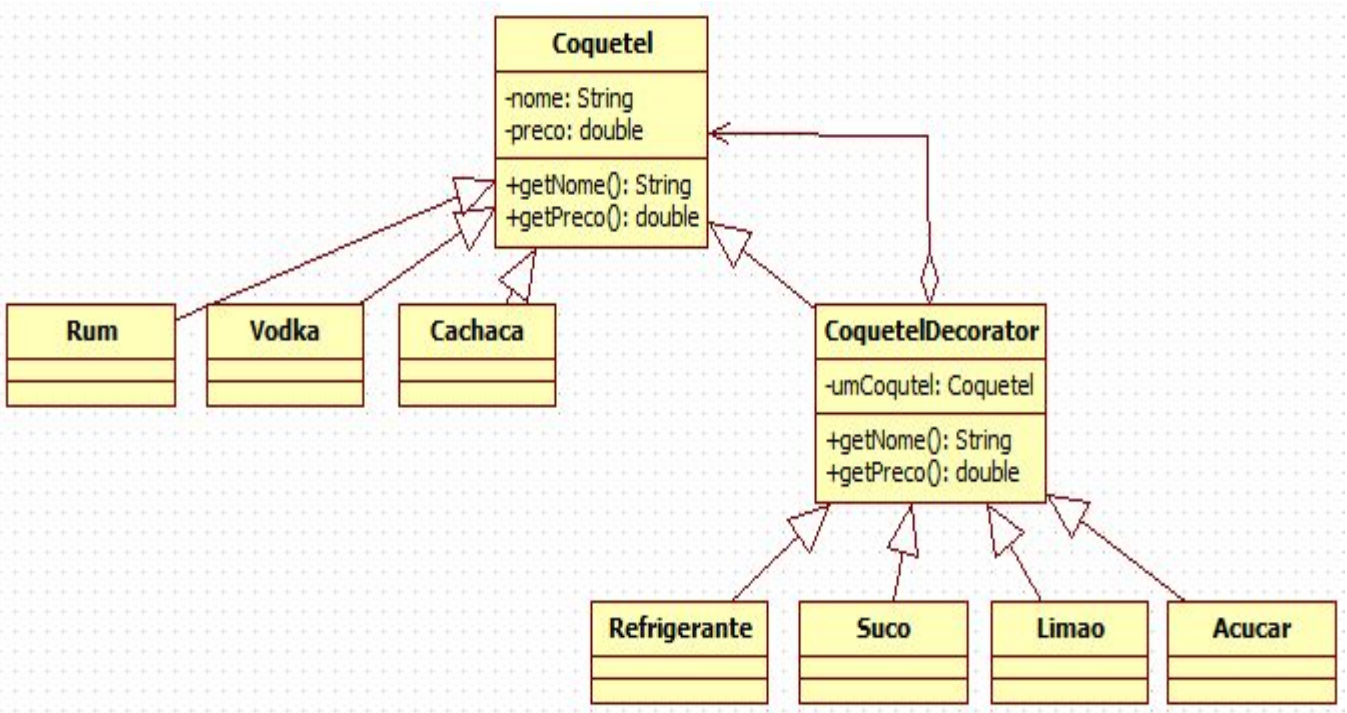
- Limão
- Refrigerante
- Suco
- Leite condensado
- Gelo
- Açúcar

Então, como possíveis coquetéis temos:

- Vodka + Suco + Gelo + Açúcar
- Tequila + Limão + Sal
- Cachaça + Leite Condensado + Açúcar + Gelo


CaipirinhaIogurte
CaipirinhaMaisLeiteCondensado
Caipirinha
Tequila
Coquetel
-nome: String
-preco: double
+getNome(): String
+getPreco(): double
CaipirinhaMaisAcucar
HiFi
IceVodkaLimao
Cuba Libre
Whisky
WhiskyMaisEnergético
Rum

“DECORANDO”


Coquetel
-nome: String
-preco: double
+getNome(): String
+getPreco(): double
Rum
Vodka
Cachaca
CoquetelDecorator
-umCoqutel: Coquetel
+getNome(): String
+getPreco(): double
Refrigerante
Suco
Limao
Acucar

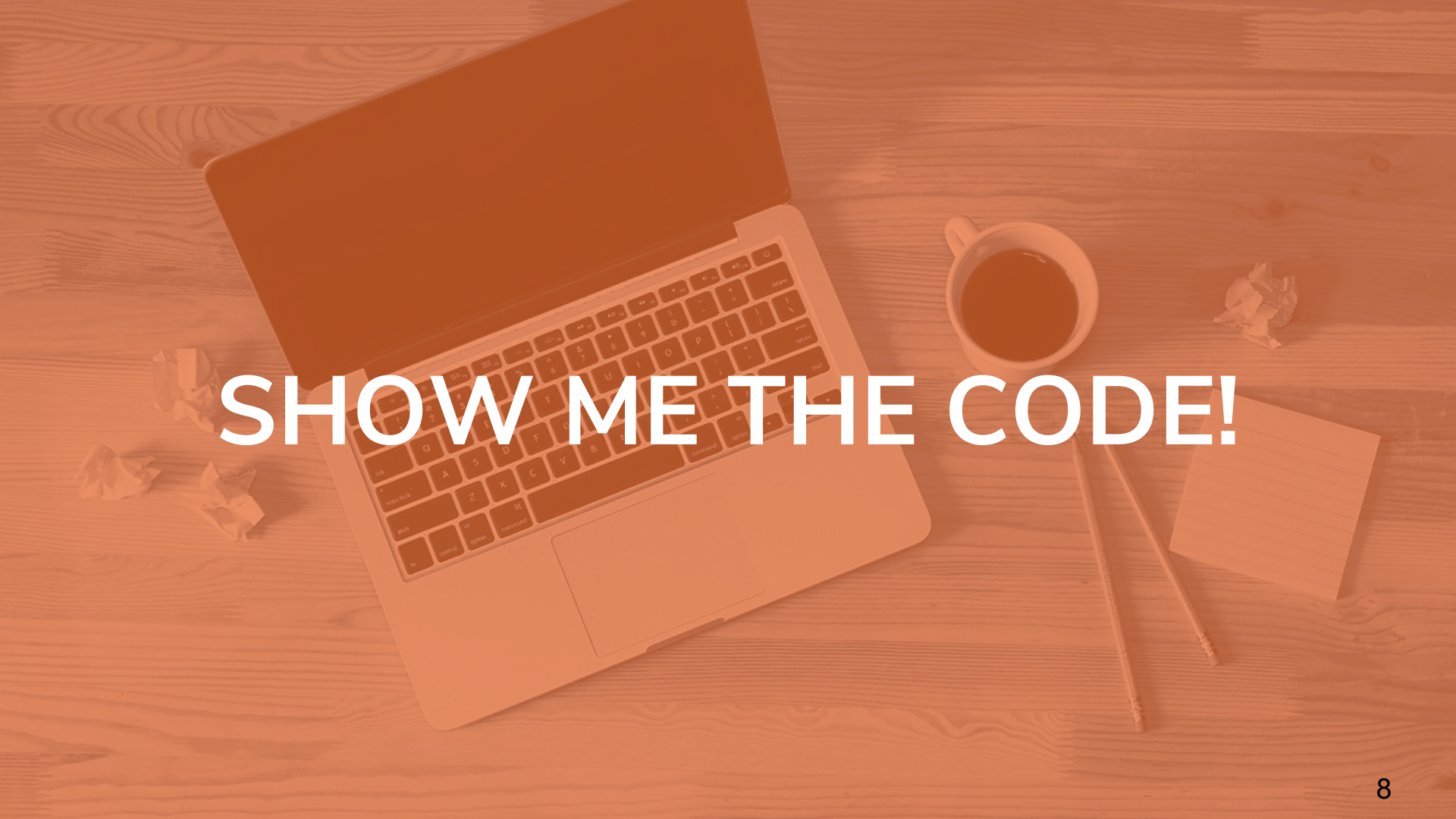
VANTAGENS X DESVANTAGENS

Vantagens:

- Maior flexibilidade do que a herança estática;
- Aplica apenas funcionalidades necessárias, evitando que objetos possuem métodos nunca utilizados.

Desvantagens:

- Comportamento altamente dinâmico, não é possível verificar se um coquetel possui um decorador Limão, por exemplo.


SHOW ME THE CODE!