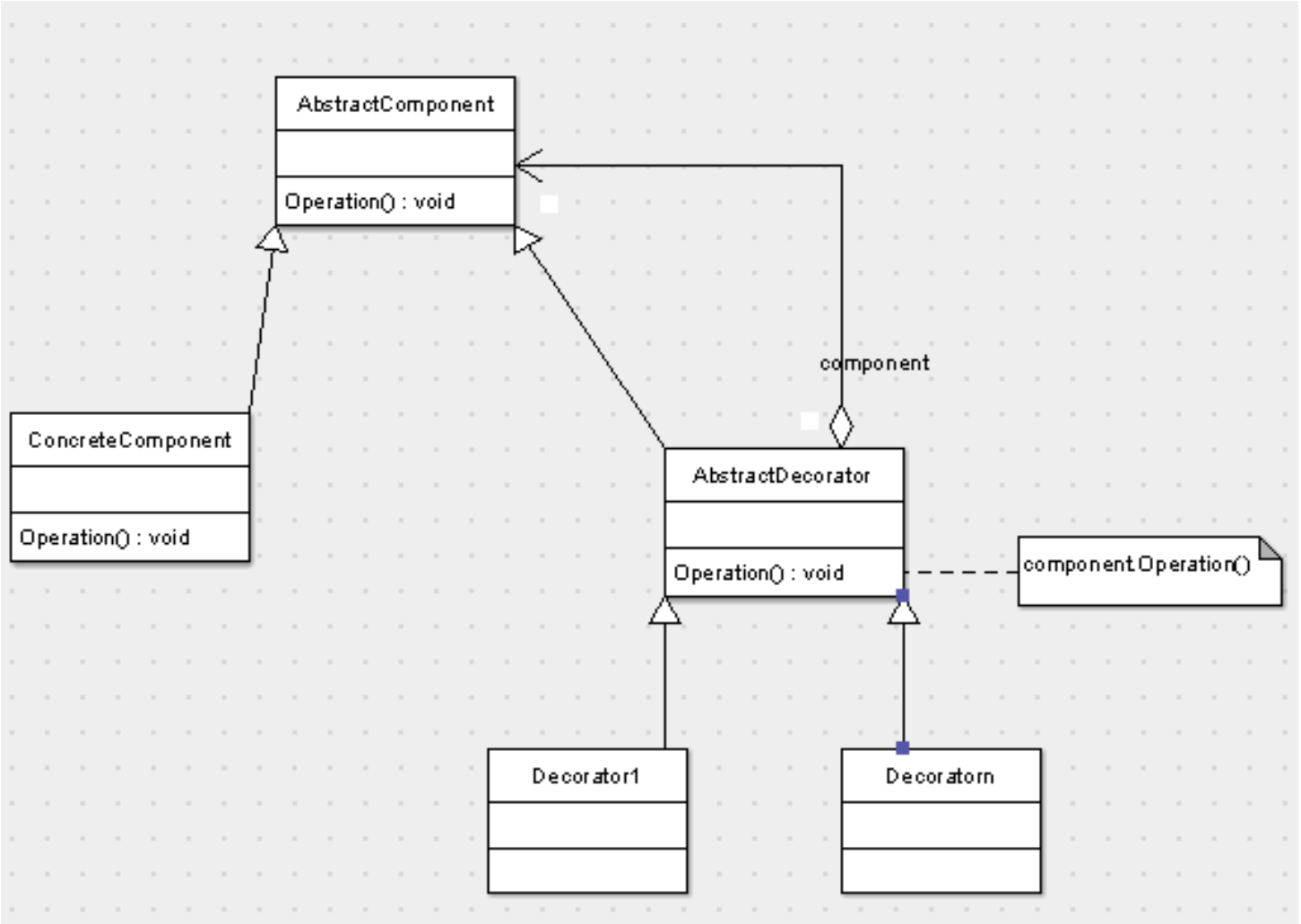
Decorator

Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

Elementos Essenciais

- **Nome:** Decorator *(Wrapper)*
- **Problema:** Podemos querer adicionar responsabilidades a objetos individuais e não a suas classes. Por exemplo, podemos querer usar um componente visual lhe acrescentando uma borda, um scroll ou qualquer outra propriedade.
- **Solução:** Dinamicamente, agregar responsabilidades adicionais a objetos. Os Decorators fornecem uma alternativa flexível ao uso de subclasses para extensão de funcionalidades.
- **Consequências:** Maior flexibilidade para acrescentar comportamentos (específico a um objeto). Não é possível, a partir do "objeto decorado" verificar quem o decora (pelo menos não a priori). O comportamento desse padrão se alinha muito com o conceito de recursão aplicado a métodos.

Padrão 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. Os coquetéis são feitos da combinação de uma bebida base e vários outros adicionais que compõe a bebida. Por exemplo:

Conjunto de bebidas:

- Cachaça
- Rum
- Vodka
- Tequila

Conjunto de adicionais:

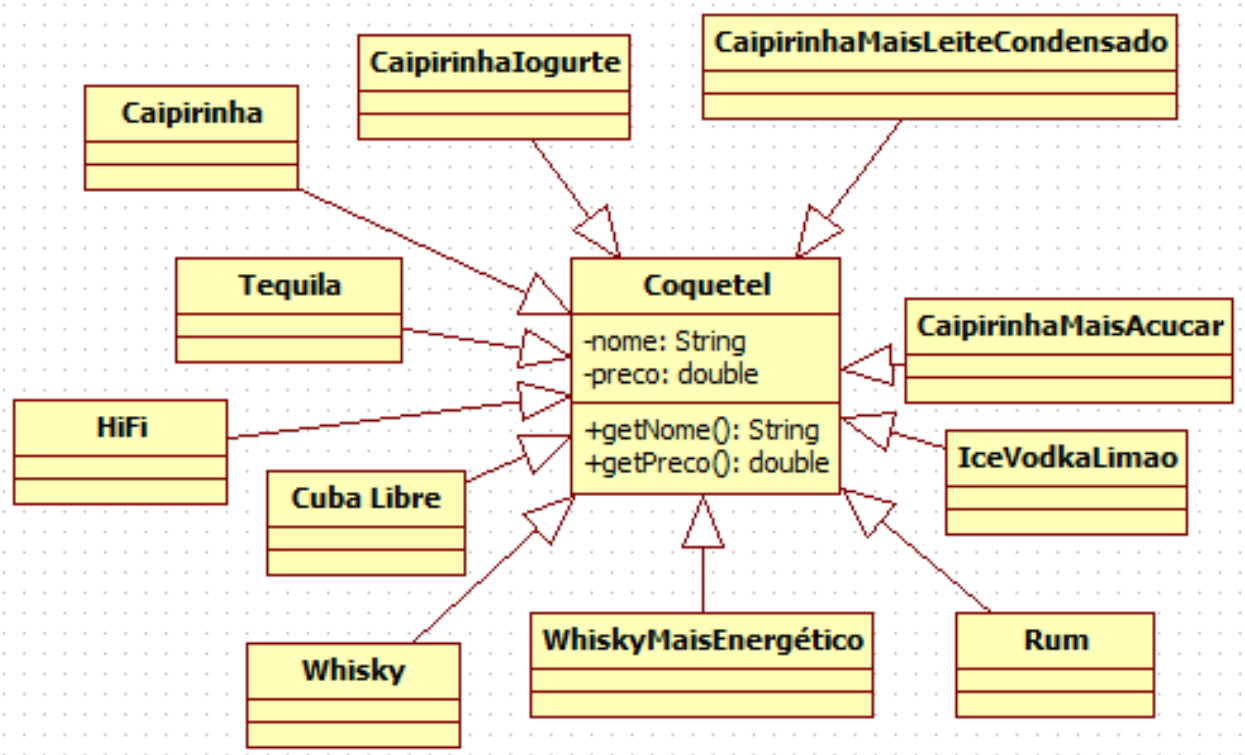
- Limão
- Refrigerante
- Suco
- Leite condensado
- Gelo
- Açúcar

Então, como possíveis coquetéis temos:

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

Fonte: https://brizeno.wordpress.com/category/padroes-de-projeto/decorator/ (acessado em 10/02/2020)

Possível solução?

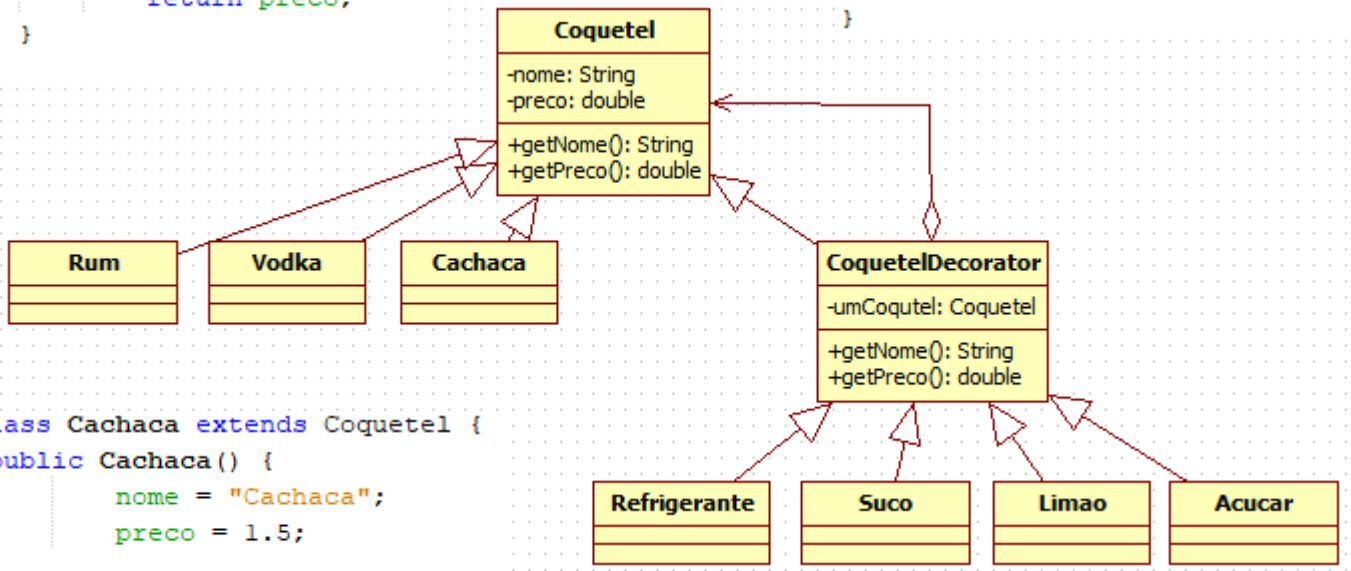


Muitas possibilidades de combinação!!!

Padrão Decorator

```
public abstract class Coquetel {
        String nome;
        double preco;
        public String getNome() {
                return nome;
        }
        public double getPreco() {
                return preco;
        }
}
```

```
public abstract class CoquetelDecorator extends Coquetel {
        Coquetel coquetel;
        public CoquetelDecorator(Coquetel umCoquetel) {
                coquetel = umCoquetel;
        }
        @Override
        public String getNome() {
                return coquetel.getNome() + " " + nome;
        }
        @Override
        public double getPreco() {
                return coquetel.getPreco() + preco;
        }
}
```




```
public class Cachaca extends Coquetel {
        public Cachaca() {
                nome = "Cachaca";
                preco = 1.5;
        }
}
```

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

Classe com método main()

```java
public class Decorator {
    public static void main(String[] args) {
        Coquetel meuCoquetel = new Rum();
        System.out.println(meuCoquetel.getNome() + " = " + meuCoquetel.getPreco());

        System.out.println(meuCoquetel instanceof Rum);

        meuCoquetel = new Refrigerante(meuCoquetel);
        System.out.println(meuCoquetel.getNome() + " = "
                + meuCoquetel.getPreco());

        meuCoquetel = new Suco(meuCoquetel);
        System.out.println(meuCoquetel.getNome() + " = "
                + meuCoquetel.getPreco());

        meuCoquetel = new Acucar(meuCoquetel);
        System.out.println(meuCoquetel.getNome() + " = "
                + meuCoquetel.getPreco());

        meuCoquetel = new Acucar(meuCoquetel);
        System.out.println(meuCoquetel.getNome() + " = "
                + meuCoquetel.getPreco());

        // Problema? Pode ser
        System.out.println(meuCoquetel instanceof Rum);
    }
}
```

```
run:
Rum = 2.5
true
Rum + Refrigerante = 3.5
Rum + Refrigerante + Suco = 4.3
Rum + Refrigerante + Suco + Acucar = 4.8
Rum + Refrigerante + Suco + Acucar + Acucar = 5.3
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Considerações sobre o padrão

- O padrão Decorator é muito comum de ser usado para acréscimo de comportamentos em componentes visuais.
- Podemos ter um campo de texto (TextView, por exemplo) com um decorator associado a ele para colocar um scroll (ScrollDecorator, por exemplo). Nesse contexto, o uso do **instanceof** não seria um problema significativo, pois provavelmente procuraríamos pela classe decorator (ScrollDecorator) e não pela classe "decorada" (TextView).
- Repare o "comportamento recursivo" nos métodos getNome e getPreco.

Dúvidas?