Padrão Decorator

Prof. Me. Rodrigo Ayres

Padrão Decorator

- 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:

Decorator

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

Decorator

- Então, como possíveis coquetéis temos:
  - Vodka + Suco + Gelo + Açúcar
  - Tequila + Limão + Sal
  - Cachaça + Leite Condensado + Açúcar + Gelo

- E então, como representar isto em um sistema computacional?
  - Bom, poderíamos utilizar como uma solução simples uma classe abstrata Coquetel extremamente genérica e, para cada tipo de coquetel construir uma classe concreta.

Decorator

- Então teríamos a classe base Coquetel:

```
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 }
```

Decorator

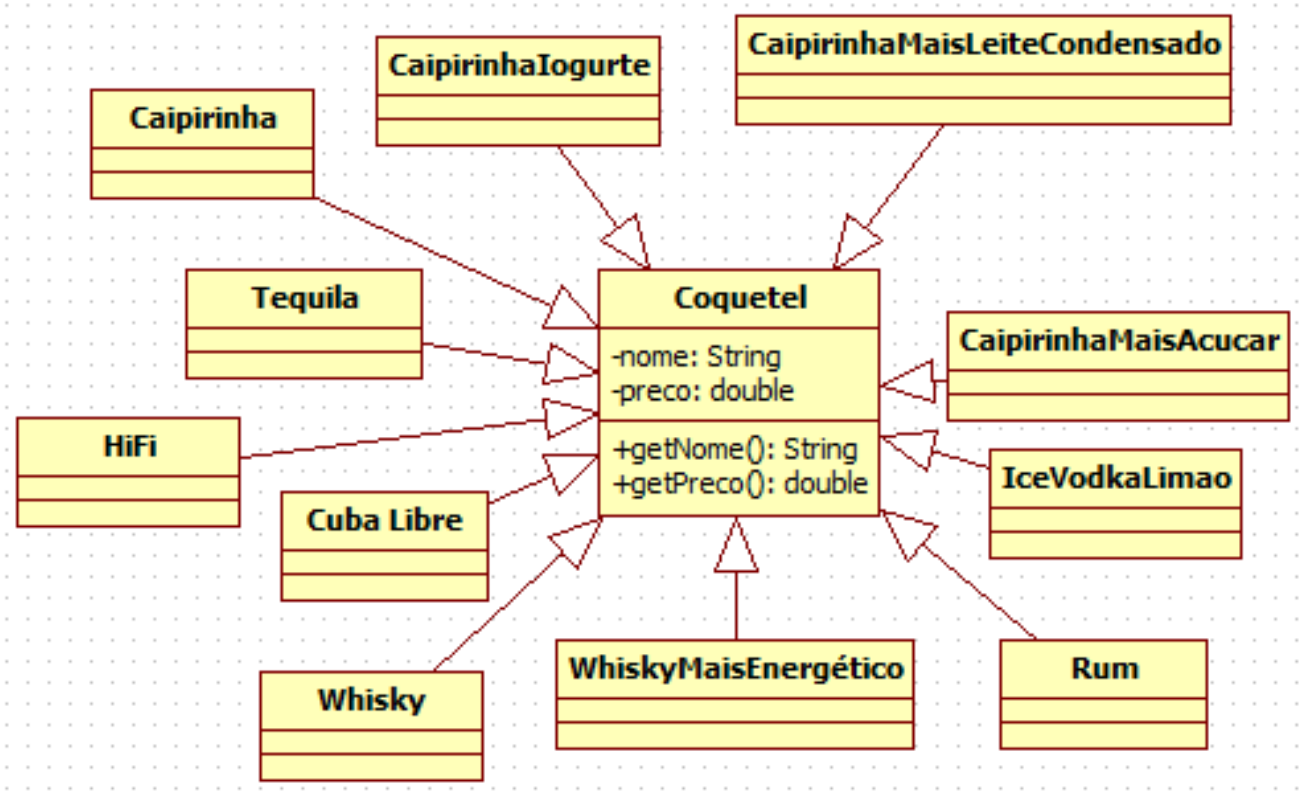
- A nossa classe define apenas o nome e o preço da bebida para facilitar a exemplificação. Uma classe coquetel concreta seria, por exemplo, a Caipirinha:

```
1 public class Caipirinha extends
Coquetel {
2     public Caipirinha() {
3         nome = "Caipirinha";
4         preco = 3.5;
5     }
6 }
```

Decorator

- No entanto, como a especialidade do bar são coquetéis, o cliente pode escolher montar seu próprio coquetel com os adicionais que ele quiser.
- De acordo com nosso modelo teríamos então que criar várias classes para prever o que um possível cliente solicitaria! Imagine agora a quantidade de combinações possíveis?
- Veja o diagrama UML abaixo para visualizar o tamanho do problema:

Decorator

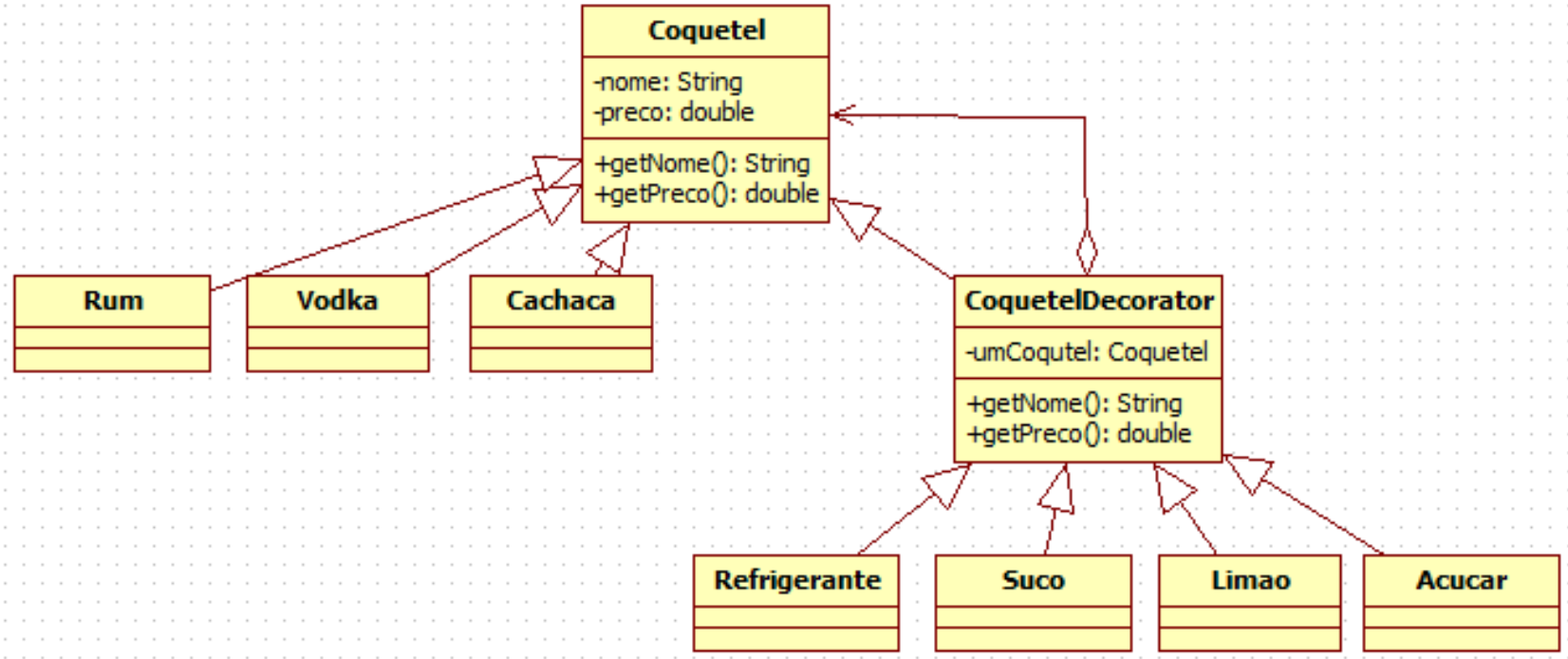

CaipirinhaIogurte
CaipirinhaMaisLeiteCondensado
Caipirinha
Tequila
Coquetel
-nome: String
-preco: double
+getNome(): String
+getPreco(): double
CaipirinhaMaisAcucar
HiFi
Cuba Libre
IceVodkaLimao
Whisky
WhiskyMaisEnergético
Rum

Decorator

- Além disso, pode ser que o cliente deseje adicionar doses extras de determinados adicionais, desse modo não seria possível modelar o sistema para prever todas as possibilidades!

- Então, como resolver o problema?

Decorator


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

Decorator

```
1  public class Cachaca extends Coquetel {
2      public Cachaca() {
3          nome = "Cachaça";
4          preco = 1.5;
5      }
6  }
```

- Certo, então todos os objetos possuem o mesmo tipo Coquetel, esta classe define o que todos os objeto possuem e é igual a classe já feita antes.
- As classes de bebidas concretas definem apenas os dados relativos a ela. Como exemplo vejamos o código da bebida Cachaça:

Decorator

- Todas as classes de bebidas possuirão a mesma estrutura, apenas definem os seus atributos.
- 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.

- Vejamos o código para exemplificar:

Decorator

```
1  public abstract class CoquetelDecorator extends
   Coquetel {
2      Coquetel coquetel;
3
4      public CoquetelDecorator(Coquetel umCoquetel) {
5          coquetel = umCoquetel;
6      }
7
8      @Override
9      public String getNome() {
10         return coquetel.getNome()  + " + " + nome;
11     }
12
13     public double getPreco() {
14         return coquetel.getPreco() + preco;
15     }
16 }
```

Decorator

- Lembre-se de que 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, vejamos o exemplo do adicional Refrigerante:

Decorator

• Perceba que no construtor do decorador é necessário passar um objeto Coquetel qualquer, este objeto pode ser tanto uma bebida quanto outro decorador. Ai está o conceito chave para o padrão Decorator. Vamos acrescentando vários decoradores em qualquer ordem em uma bebida. Vamos ver agora como o padrão seria utilizado, veja o seguinte código do método main:

```
1 public class Refrigerante extends CoquetelDecorator
2 {
3
4     public Refrigerante(Coquetel umCoquetel) {
5         super(umCoquetel);
6         nome = "Refrigerante";
7         preco = 1.0;
8     }
9 }
```

Decorator

```
1 public static void main(String[] args) {
2       Coquetel meuCoquetel = new Cachaca();
3       System.out.println(meuCoquetel.getNome() + " = "
4             + meuCoquetel.getPreco());
5
6       meuCoquetel = new Refrigerante(meuCoquetel);
7       System.out.println(meuCoquetel.getNome() + " = "
8             + meuCoquetel.getPreco());
9   }
```