Padrões de Projeto

Decorator

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação

Agenda

Bem-vindo ao Starbuxx Coffee

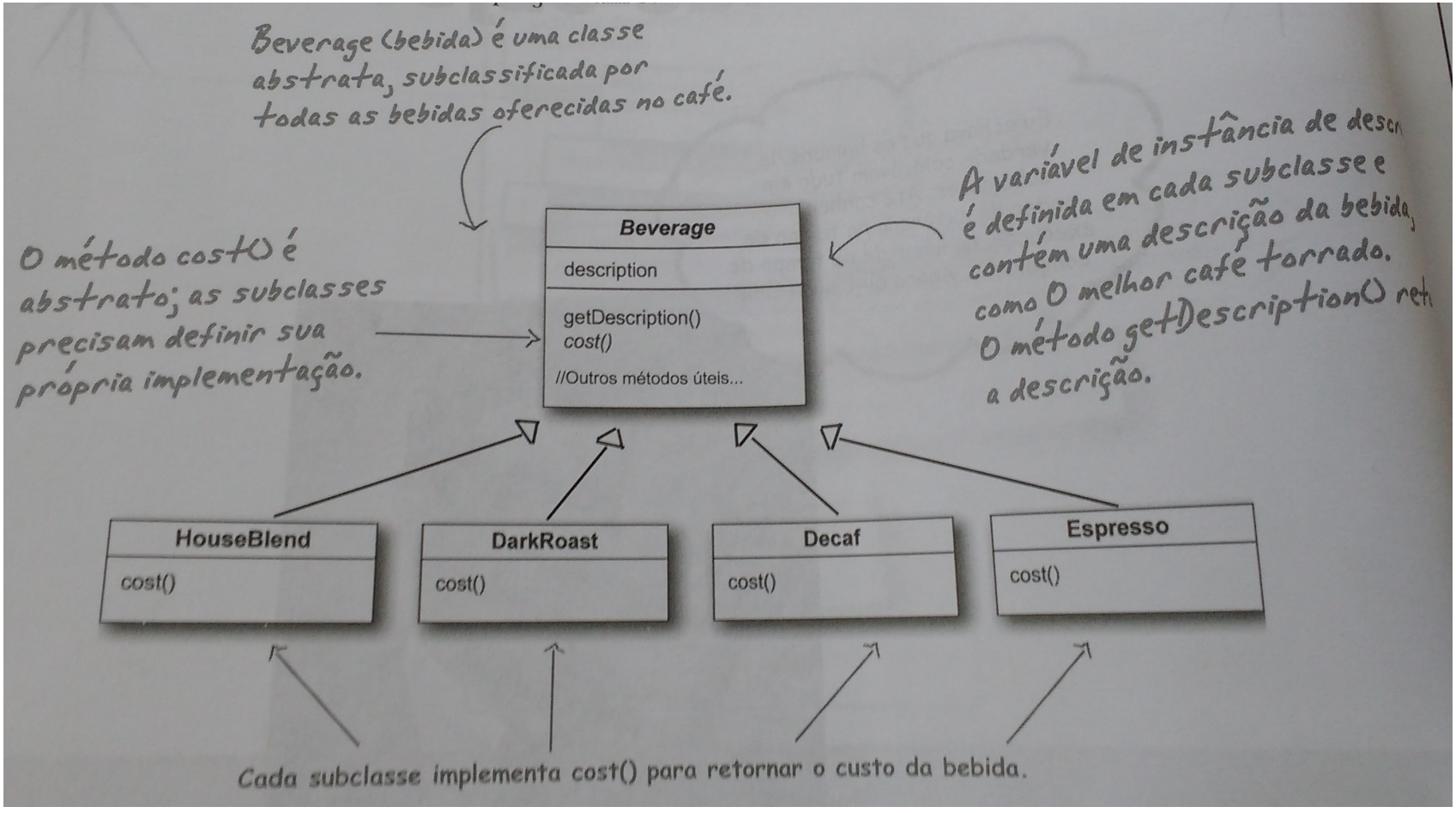
O Starbuzz Coffee (empresa como uma lanchonete) ficou conhecido como o café com o crescimento mais rápido da região.

Se você já viu um na sua esquina, olhe para o outro lado da rua: você verá outro.

Como eles cresceram muito rápido, estão tendo dificuldades em atualizar seus sistemas de pedidos para corresponder a suas ofertas de bebidas.

Bem-vindo ao Starbuzz Coffee

Quando eles entraram no negócio, projetaram suas classes assim....

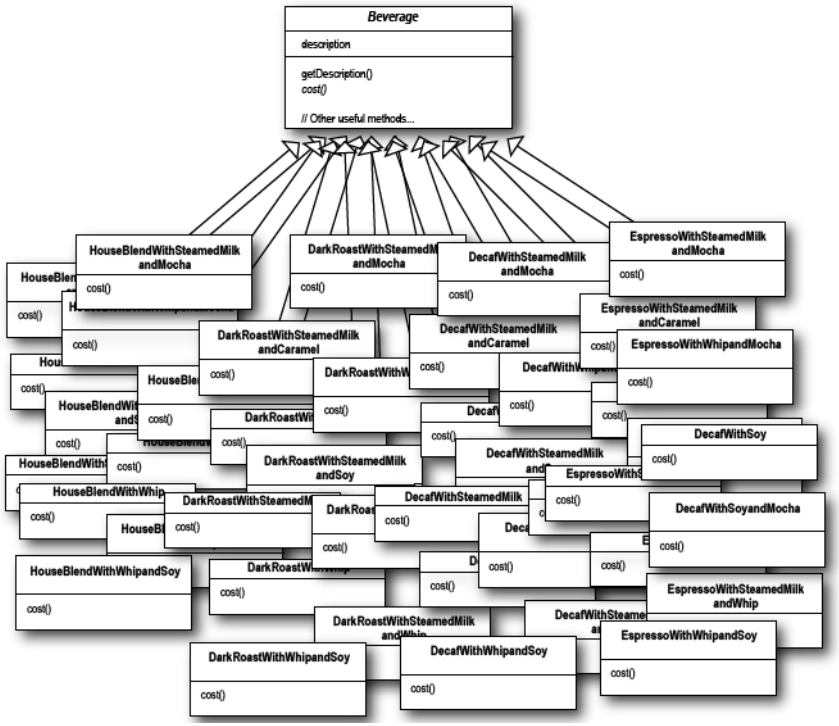


Bem-vindo ao Starbuzz Coffee

Além do café, você também pode pedir vários condimentos, como leite com espumas, soja e moca (também conhecido como chocolate), misturados com leite batido.

Como o Starbuzz cobra um valor por cada condimento, eles realmente precisam incluí-los no sistema de pedido.

Bem-vindo ao Starbuzz Coffee



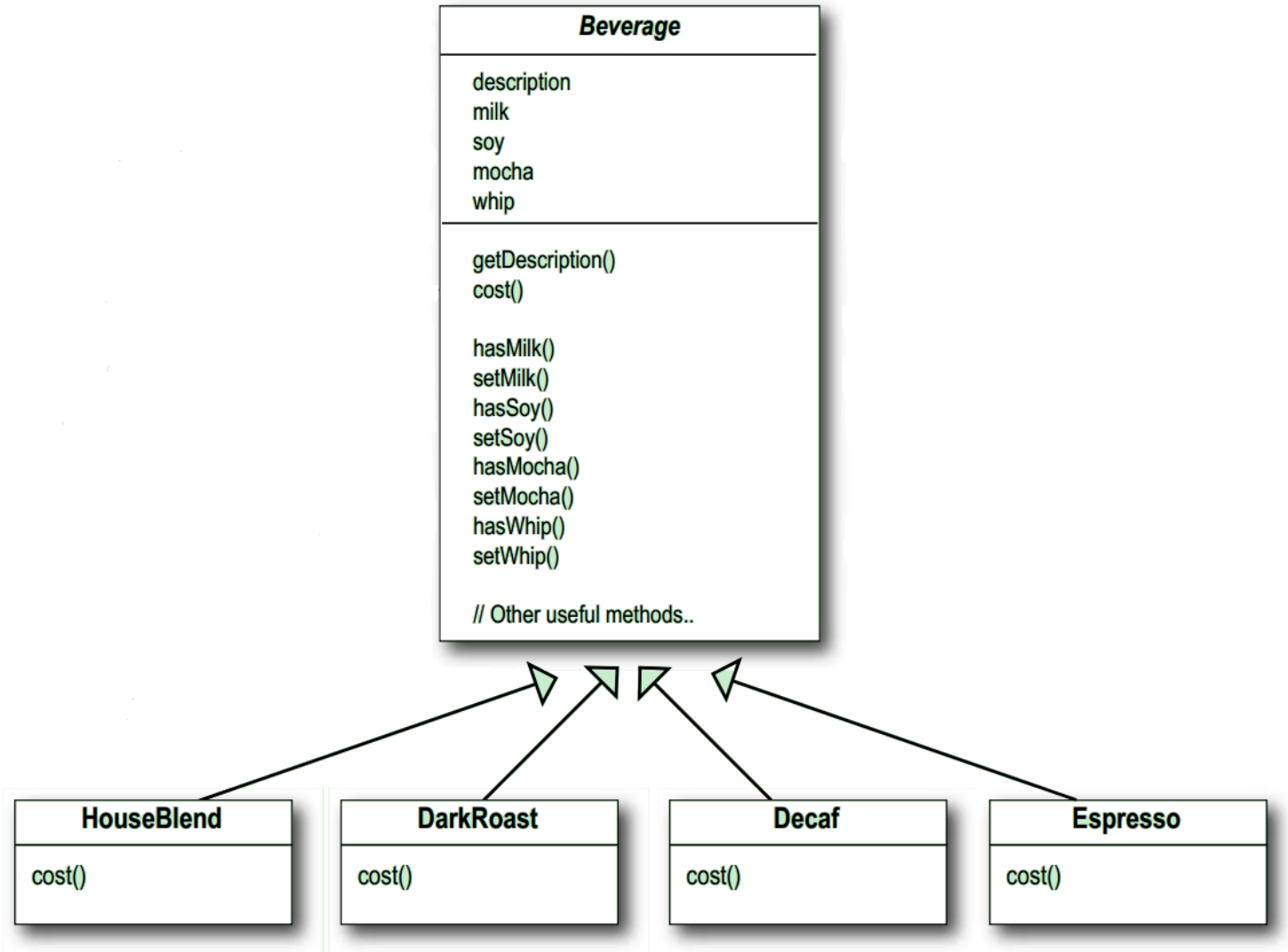
Uau!! Houve uma explosão de classes...

Bem-vindo ao Starbuzz Coffee

Comentário de um Programador...

Isto é estúpido! Por que precisamos de todas essas classes? Não podemos usar apenas variáveis de instância e herança na superclasse para monitorar os condimentos?

Bem-vindo ao Starbuzz Coffee



Bem-vindo ao Starbuzz Coffee

Comentário do mesmo Programador...

Certo, um total de 5 classes. Esse definitivamente é o caminho.

Comentário do Gerente do Projeto

Não tenho tanta certeza. Posso ver alguns possíveis problemas com essa abordagem em como o design poderia precisar mudar no futuro.

Bem-vindo ao Starbuzz Coffee

Problemas dessa Abordagem

- Mudanças de preços dos condimentos irão nos forçar a alterar o código existente
- Novos condimentos irão nos forçar a adicionar novos métodos e alterar o método de custo na superclasse
- Agora podemos ter novas bebidas. Para algumas dessas bebidas (chá gelada?!) os condimentos podem não ser apropriados, mas a subclasse (Tea) ainda irá herdar os métodos como hasWhip()
- E se o cliente quiser uma moca dupla?

Bem-vindo ao Starbuzz Coffee

Ok, vimos que representar nosso esquema de preço de bebida mais condimentos com herança não funciona muito bem.

- Obtemos explosões de classes
- Projetos rígidos
- ou adicionamos funcionalidades à base que não é apropriada para algumas subclasses

Então, isto é o que faremos:

Vamos começar com uma bebida e "decorá-la" com os condimentos no tempo de execução.

Princípio de Projeto

Classes devem ser fechadas para modificação e abertas para extensão

Bem-vindo ao Starbuzz Coffee

Exemplo:

1. Começamos com o nosso objeto DarkRoast
2. O cliente quer Mocha, então criamos um objeto Mocha e englobamos DarkRoast nele
3. O cliente também quer Whip, então criamos um decorador Whip e colocamos Mocha

Bem-vindo ao Starbuzz Coffee

Agora, é hora de computar o custo do cliente.

Fazemos isso chamando cost() no decorador externo, Whip, e Whip vai delegar o cálculo do custo para o objeto que decora.

Depois que obtiver um custo, ele irá adicionar o custo de Whip.

Bem-vindo ao Starbuzz Coffee

Bem-vindo ao Starbuzz Coffee

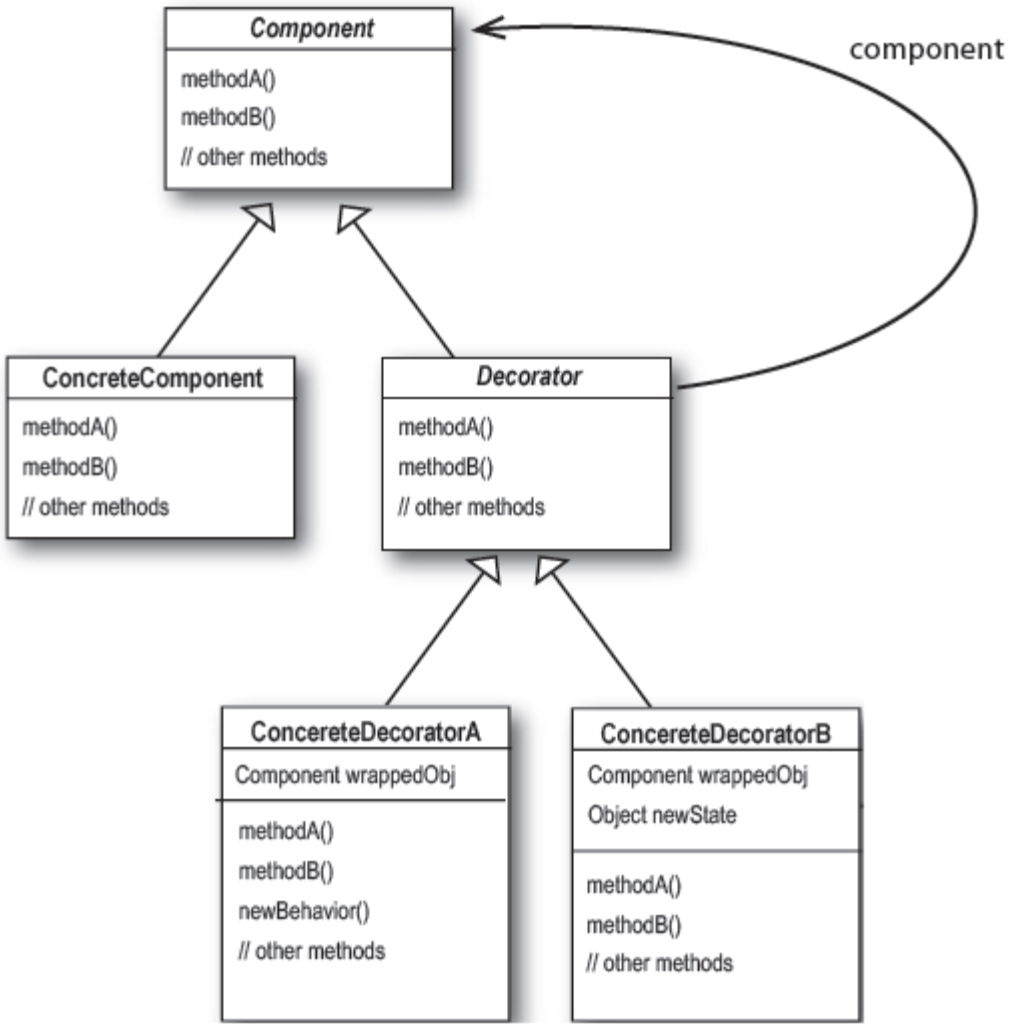
Agora, vamos ver como isso tudo realmente funciona analisando a definição do Padrão Decorador e escrevendo alguns códigos.

Padrão Decorator

O Padrão Decorator anexa responsabilidades adicionais a um objeto dinamicamente. Os decoradores fornecem uma alternativa flexível de subclasse para estender a funcionalidade.

Quando há herança de um comportamento por meio de subclasses, este comportamento é definido estaticamente em tempo de compilação. Contudo, por meio da composição é possível estender o comportamento do objeto de forma dinâmica e em tempo de execução.

Padrão Decorator

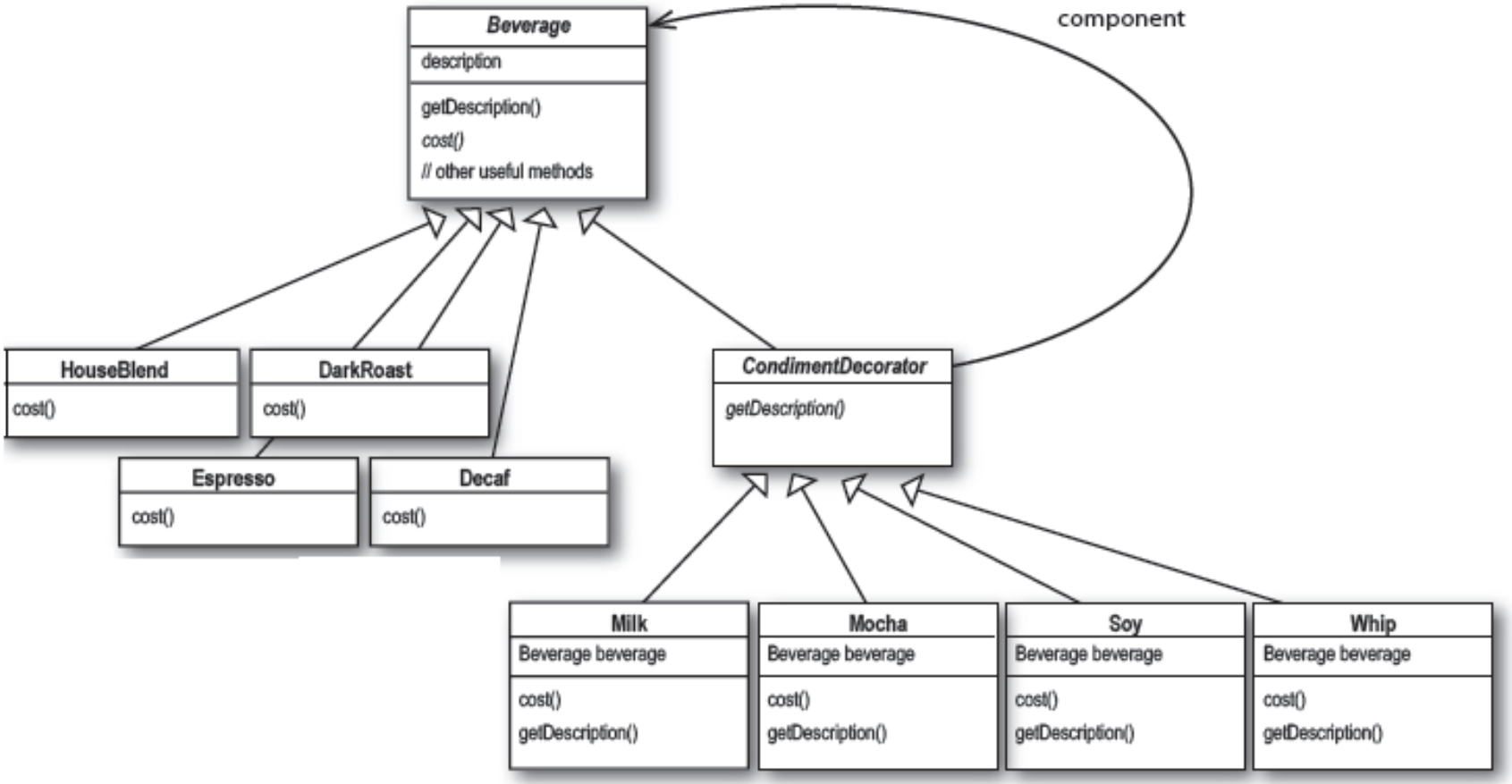


Padrão Decorator

- **Component**: Define a interface para os componentes onde teremos decorators.
- **ConcreteComponent**: Representam a implementação concreta do componente que poderá ser decorado posteriormente.
- **Decorator**: Representa a particularidade do componente decorator manter a referência para o componente decorado. **Pode ser omitida, como no exemplo apresentado.**
- **ConcreteDecorator:** Representam a implementação dos decorators.

Padrão Decorator

Decorator para o Problema Starbuzz



Padrão Decorator

- Beverage é uma classe abstrata de componente. Cada bebida pode ser usada sozinha ou englobada por um decorador;
  - *Expresso*, *DarkRoast* e *HouseBlend* são componentes concretos que estendem de *Beverage* no que caracteriza que essas classes são um tipo de bebida;
  - Esses objetos receberão dinamicamente novos comportamentos;

Padrão Decorator

- Beverage é uma classe abstrata de componente. Cada bebida pode ser usada sozinha ou englobada por um decorador;
  - *Expresso*, *DarkRoast* e *HouseBlend* são componentes concretos que estendem de *Beverage* no que caracteriza que essas classes são um tipo de bebida;
  - Esses objetos receberão dinamicamente novos comportamentos;
- *CondimentDecorator* "tem-uma"(engloba) uma bebida, o que significa que o *CondimentDecorator* tem uma variável de instância que contém uma referência a uma bebida

Padrão Decorator

- Beverage é uma classe abstrata de componente. Cada bebida pode ser usada sozinha ou englobada por um decorador;
  - *Expresso*, *DarkRoast* e *HouseBlend* são componentes concretos que estendem de *Beverage* no que caracteriza que essas classes são um tipo de bebida;
  - Esses objetos receberão dinamicamente novos comportamentos;
- *CondimentDecorator* "tem-uma"(engloba) uma bebida, o que significa que o *CondimentDecorator* tem uma variável de instância que contém uma referência a uma bebida
- Os decoradores podem adicionar novos métodos; no entanto, o novo comportamento geralmente é adicionado fazendo cálculo antes e depois de um método existente no componente.

Padrão Decorator

- Beverage é uma classe abstrata de componente. Cada bebida pode ser usada sozinha ou englobada por um decorador;
  - *Expresso*, *DarkRoast* e *HouseBlend* são componentes concretos que estendem de *Beverage* no que caracteriza que essas classes são um tipo de bebida;
  - Esses objetos receberão dinamicamente novos comportamentos;
- *CondimentDecorator* "tem-uma"(engloba) uma bebida, o que significa que o *CondimentDecorator* tem uma variável de instância que contém uma referência a uma bebida
- Os decoradores podem adicionar novos métodos; no entanto, o novo comportamento geralmente é adicionado fazendo cálculo antes e depois de um método existente no componente.
- Eles precisam implementar não somente *cost()*, mas também *getDescription()*

Exemplo

```
public class StarbuzzCoffee{
    public static void main(String args[]){
        Beverage beverage = new Expresso();
        System.out.println(beverage.getDescription()+" R$"+beverage.cost());

        Beverage beverage2 = new DarkRoast();
        beverage2 = new Mocha(beverage2);
        beverage2 = new Soy(beverage2);
        beverage2 = new Whip(beverage2);
        System.out.println(beverage2.getDescription()+" R$"+beverage2.cost());

        Beverage beverage3 = new HouseBlend();
        beverage3 = new Mocha(beverage3);
        System.out.println(beverage3.getDescription()+" R$"+beverage3.cost());

    }
}
```

Padrões de Projeto

Decorator

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação