Instituto de computação

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrão Decorator

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Será que herança resolver todos os problemas?

Como evitar uma explosão de subclasses?

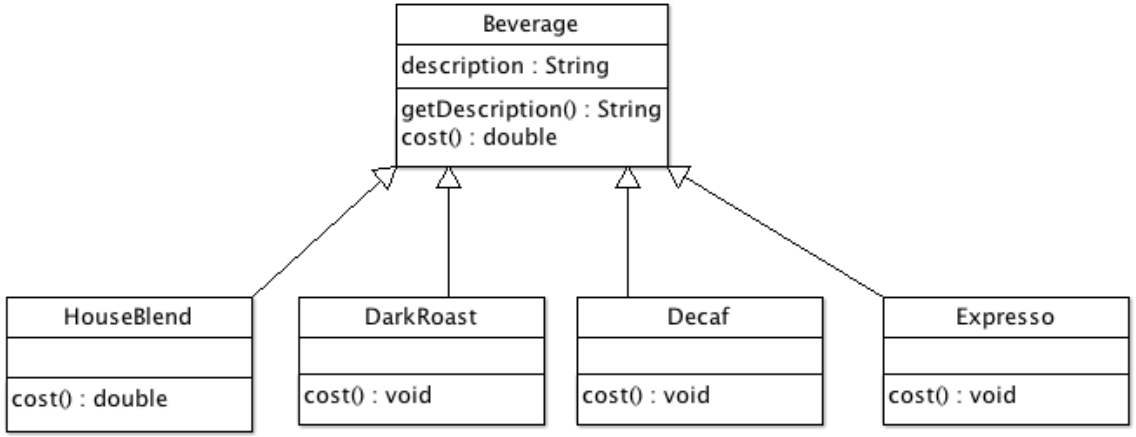
É possível adicionar responsabilidades aos objetos sem introduzir mudanças nas classes base?

Padrões de Projeto

Problema exemplo

Considere uma cafeteria que possui alguns produtos em seu menu de pedidos

Inicialmente, havia poucos produtos básicos e o sistema previa somente as classes abaixo



Padrões de Projeto

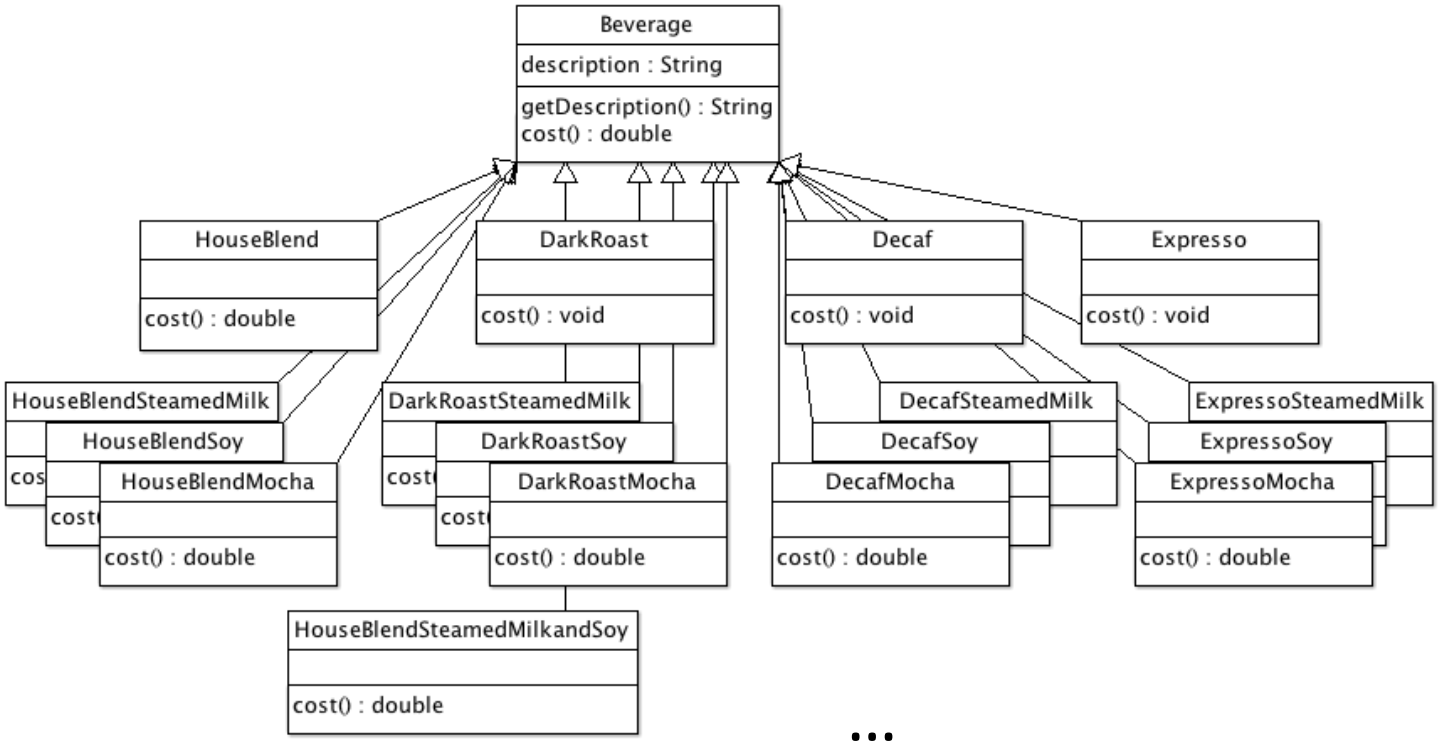
Problema exemplo

Por outro lado, a cafeteria oferecia aos clientes a possibilidade de adicionar complementos (leite, chocolate, soja, etc..), que obviamente são cobrados...

Então para manter todas essas possibilidades de combinação temos uma verdadeira explosão de classes

Padrões de Projeto

Uma explosão de classes



Vamos eliminar todas essas classes

Vamos criar variáveis de instância que indicam para cada instância de bebida, que tipo de complementos ela possui

Padrões de Projeto

Problema exemplo

Primeira solução:

Vamos acabar com todas essas classes

Vamos criar variáveis de instância que indicam para cada instância de bebida, que tipo de complementos ela possui

Padrões de Projeto

Problema exemplo

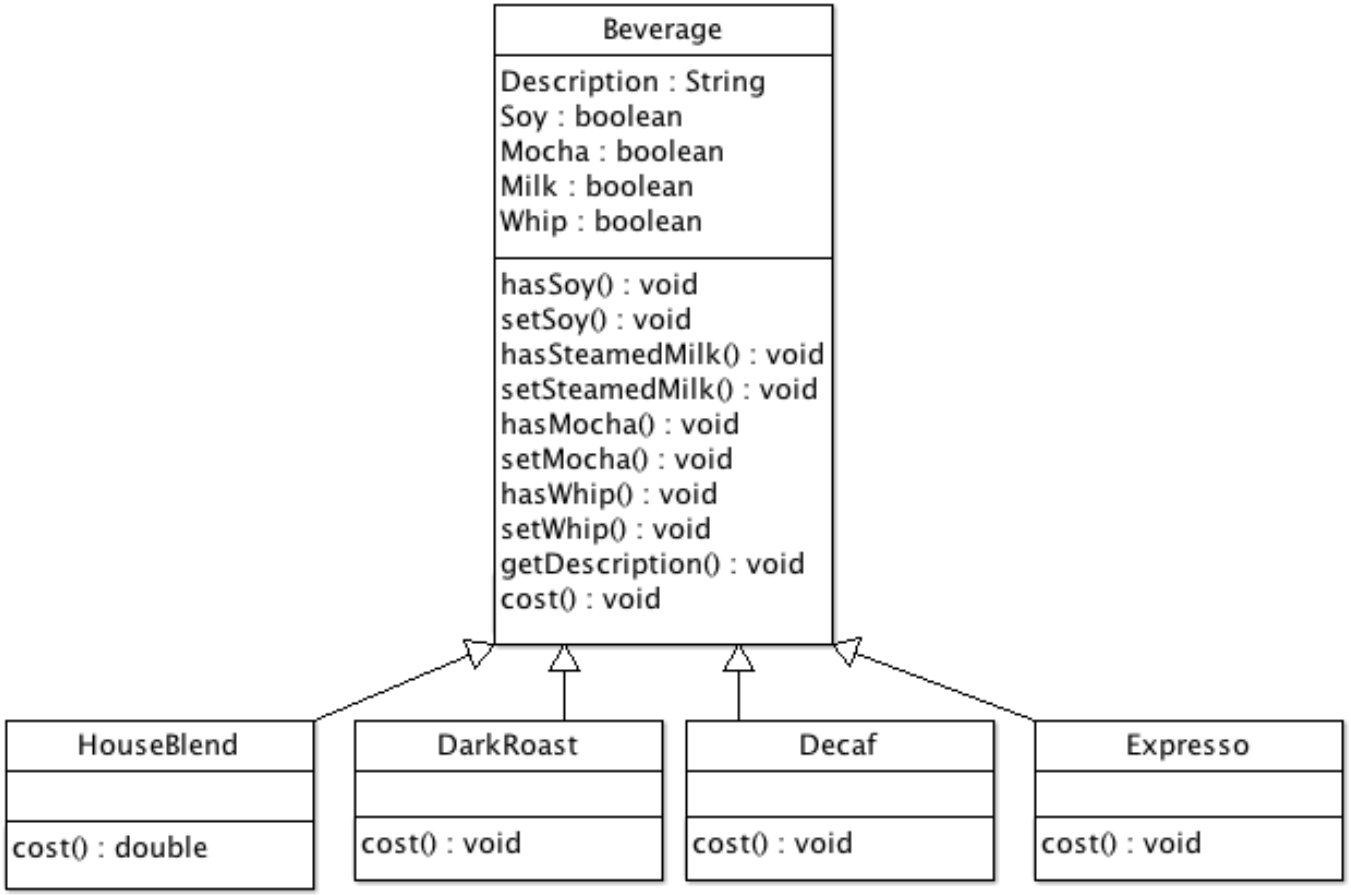
Primeira solução:

Vamos acabar com todas essas classes

Vamos criar variáveis de instância que indicam para cada instância de bebida, que tipo de complementos ela possui

Padrões de Projeto

Primeira Solução



Problemas?

Padrões de Projeto

Problema exemplo

Mudança de preços dos complementos forçarão a mudar o código existente

Novos complementos nos forçarão a adicionar novos métodos e alterar o método cost() na superclasse

Para novas bebidas alguns complementos podem não fazer sentido nenhum (Já vimos isso acontecer no exemplo do Padrão Strategy)

O que fazer para contemplar bebidas com complementos duplos? Café com mocha dupla???

Padrões de Projeto

Uso de princípios

Quarto princípio de projeto:

> Classes devem ser fechadas para modificação e abertas para extensão.

Observação sobre o princípio aberto/fechado

O uso excessivo do princípio fechado para modificação e aberto pra extensão pode levar a um código excessivamente abstrato e complexo

O princípio deve ser utilizado apenas em pontos do código em que mudanças são improváveis

Padrões de Projeto

Uso de princípios

Objetivo: permitir que classes sejam estendidas para incorporar novos comportamentos sem modificar o código existente

Qual o resultado?

> Os projetos se tornam resilientes a mudanças e flexíveis o suficiente para adquirir novas funcionalidades que satisfazem os requisitos

Padrões de Projeto

Metáfora da decoração

Como vamos resolver o problema?

Resposta: vamos usar uma metáfora de decoração




Exemplo:

Pegar um objeto DarkRoast

Decorá-lo com um objeto Mocha

Decorá-lo com um objeto Whip

Chamar o método cost() e lançar mão de delegação para adicionar o custo dos complementos

Padrões de Projeto

Metáfora da decoração

Que tipo de relacionamento dever ser estabelecido entre os objetos quando pensamos na ação decorar?

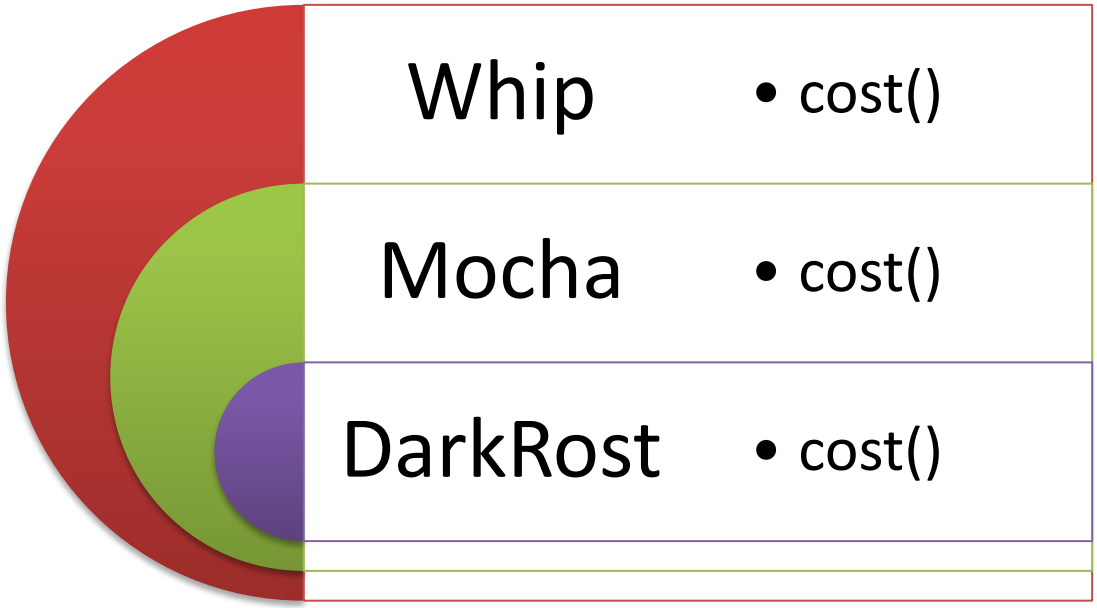
Como implementar a noção de decoração?

Como delegar as responsabilidade de adicionar os custos para a classe decorada?

Padrões de Projeto

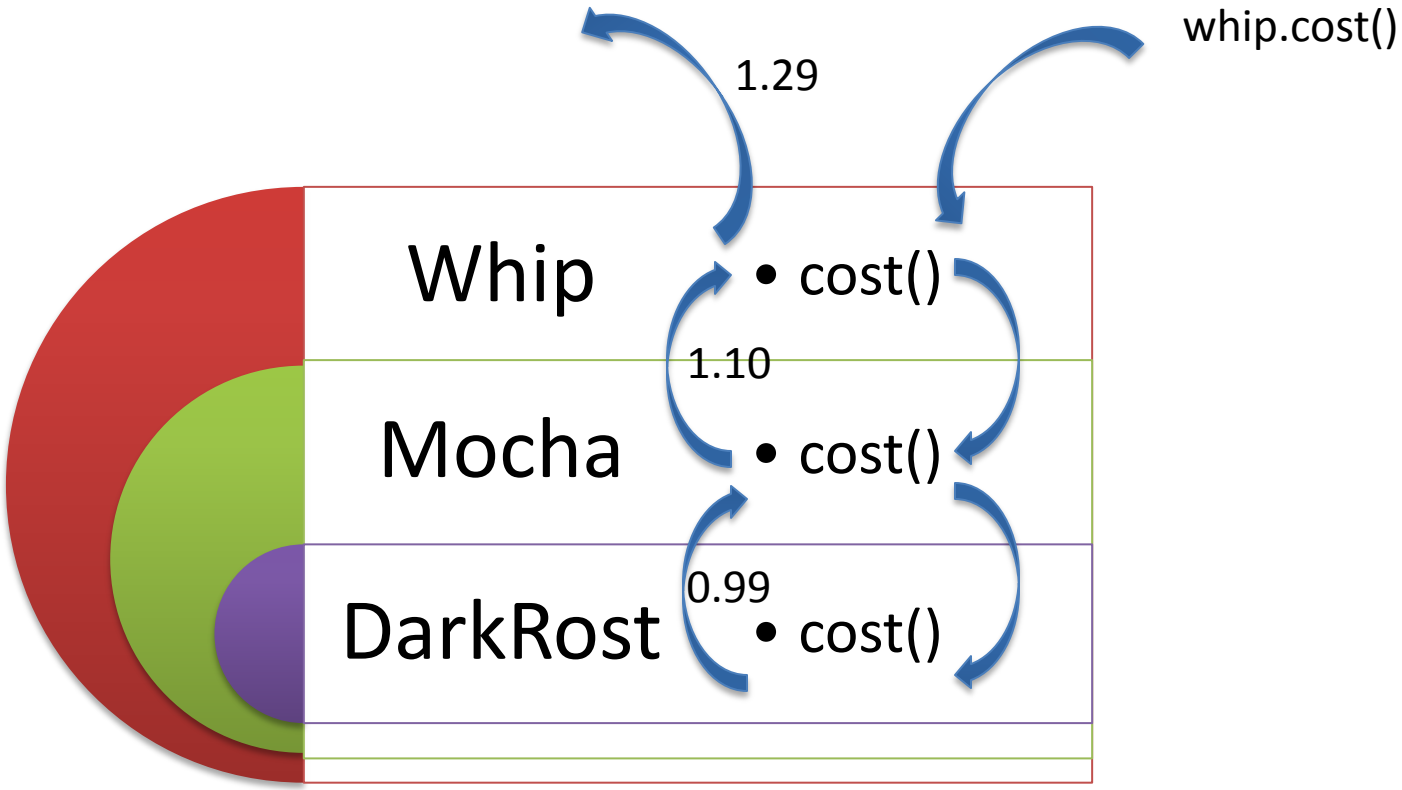
Metáfora da decoração

Vamos usar a noção de objeto wrapper (empacotador)



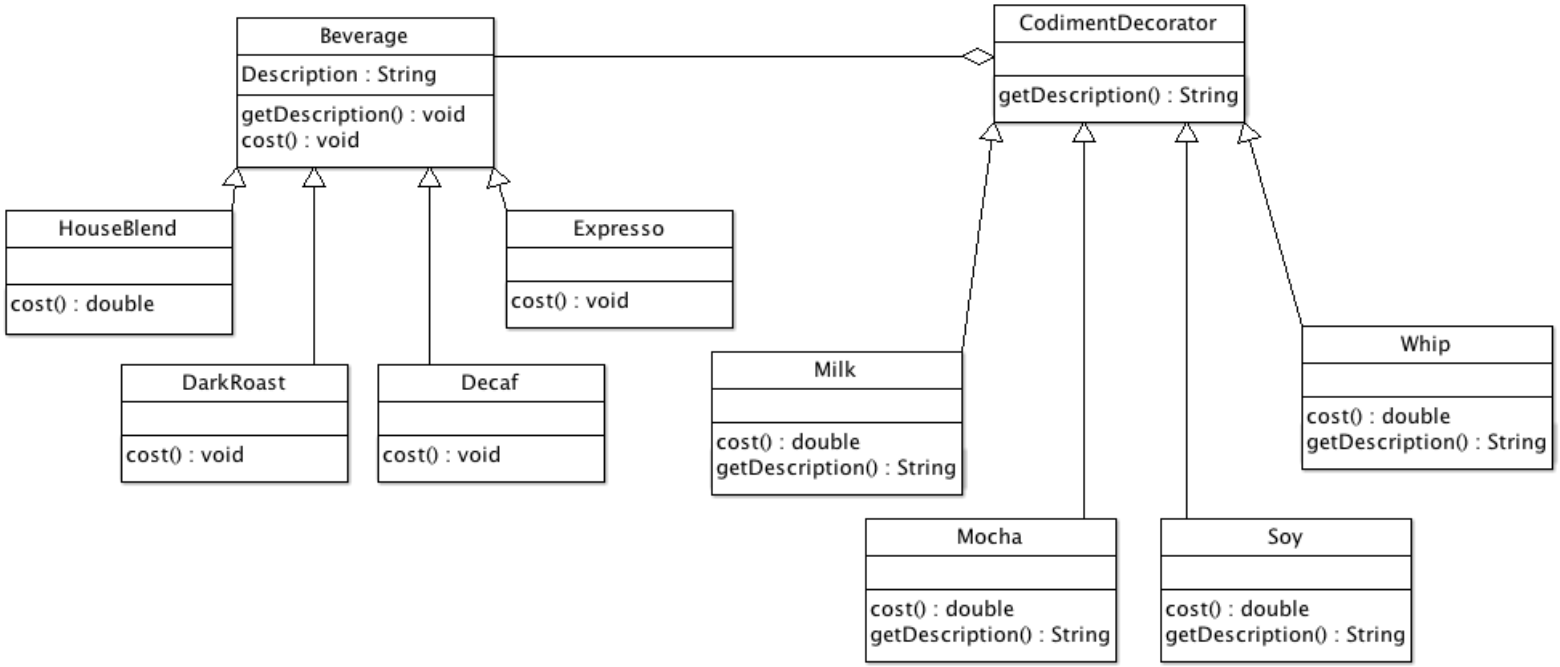
Padrões de Projeto

Metáfora da decoração



Padrões de Projeto

Segunda Solução



O Padrão Decorator anexa responsabilidades adicionais a um objeto dinâmicamente. Além disso, provê uma alternativa flexivel a subclasses para extensão de funcionalidades.

Padrões de Projeto

Padrão Decorator - características

Decorators tem o mesmo supertipo que os objetos decorados

É possível usar mais de um decorador para empacotar um objeto

Com o Padrão Decorator é possível manipular o objeto decorado no lugar do original pois eles tem o mesmo tipo

O decorador adiciona seu próprio comportamento antes ou depois de delegar o resto da tarefa para o objeto

Objetos podem ser decorados dinamicamente

Utilização

Quando se desejar adicionar responsabilidades a objetos individuais e não a uma classe inteira

Quando se desejar retirar responsabilidades

Quando extensões por subclasses for impraticável devido à grande quantidade de classes

Utilização

Quando se desejar adicionar responsabilidades a objetos individuais e não a uma classe inteira

Quando se desejar retirar responsabilidades

Quando extensões por subclasses for impraticável devido à grande quantidade de classes

Participantes

Componente – define a interface para objetos que podem ter funcionalidades adicionadas dinamicamente.

Componente concreto – define a implementação da interface do componente

Decorador – mantém uma referência para um componente concreto e segue a interface do componente

Decorador concreto – define a implementação do decorador

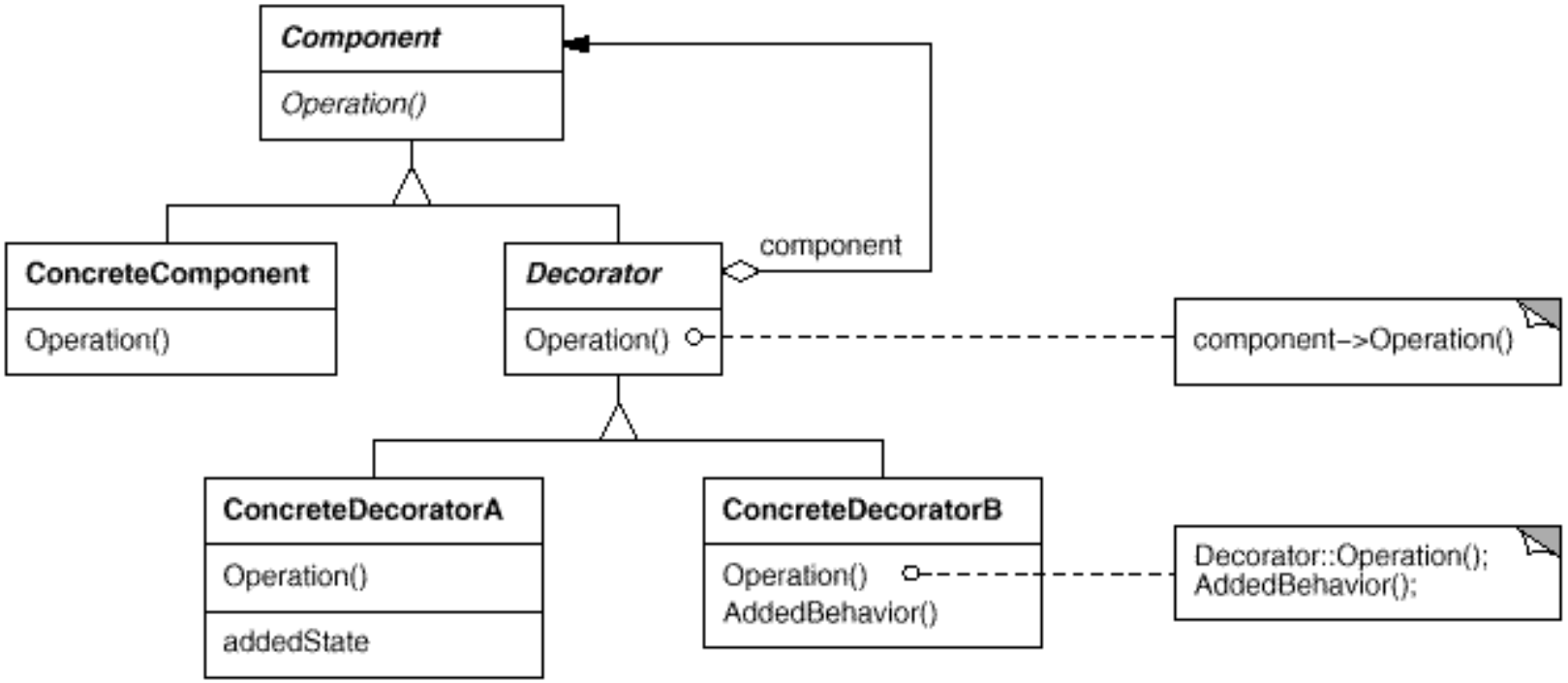
Desvantagens do padrão decorator

Leva a criação de um número muito grande de objetos pequenos semelhantes, o que pode tornar a manutenção complicada

Os Decoradores e os objetos encapsulados não são idênticos o que pode levar a falha de testes dependentes de instanceOf.

Padrões de Projeto

Usos do Padrão Decorator



Padrões de Projeto

Solução com o Padrão Decorator

```
public abstract class Beverage {

  String description = "Unknown Beverage";

  public String getDescription() {
     return description;
  }

  public abstract double cost();
}
```

```
public abstract class CondimentDecorator
extends Beverage {

  public abstract String getDescription();

}
```

```
public class Espresso extends Beverage {

public Espresso() {
    description = "Espresso";
  }

  public double cost() {
     return 1.99;
  }
}
```

```
public class HouseBlend extends Beverage{

 public HouseBlend()
    description = "House Blend Coffee";
  }

  public double cost() {
     return .89;
  }
}
```

Padrões de Projeto

Solução com o Padrão Decorator

```
public class Mocha extends CondimentDecorator {
  Beverage beverage;

  public Mocha(Beverage beverage) {
    this.beverage = beverage;
  }

  public String getDescription() {
   return beverage.getDescription() + ", Mocha";
  }

  public double cost() {
    return .20 + beverage.cost();
  }

}
```

```
public class SteamedMilk extends CondimentDecorator {
  Beverage beverage;

  public SteamedMilk(Beverage beverage) {
    this.beverage = beverage;
  }

  public String getDescription() {
   return beverage.getDescription() + ", SteamedMilk";
  }

  public double cost() {
    return .10 + beverage.cost();
  }

}
```

Algum problema?

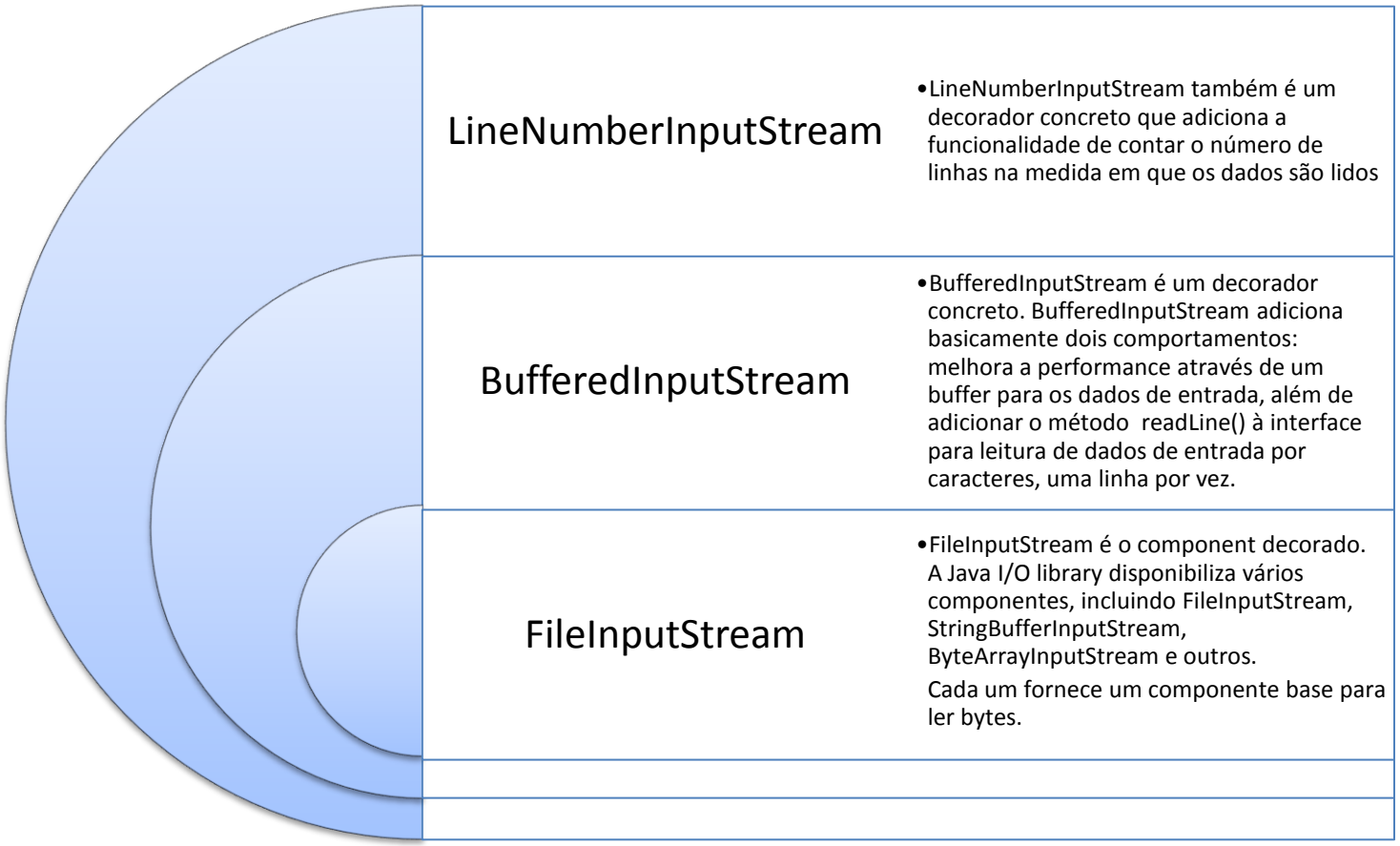
Padrões de Projeto

Solução com o Padrão Decorator

```
public class StarbuzzCoffee {

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

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

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

  }
}
```

Padrões de Projeto

Exemplo de Padrão Decorator no Java



Padrões de Projeto

Exercício I

Considere o problema da cafeteria e a solução proposta através de decoradores. Adapte a solução para que descontos sejam dados sobre os complementos a partir do segundo complemento. O segundo complemento deve ter um desconto de 25%, o terceiro 40% e os demais complementos 50%.

Padrões de Projeto

Exercício II

Escreva um método que imprima o nome do produto vendido de modo que complementos idênticos apareçam sequencialmente no tipo do produto vendido.

Padrões de Projeto

Exercício III

Escreva um decorador que converta todos os caracteres em maiúsculas para minúsculas no input stream.

Padrões de Projeto

Solução do Exercício

```
public class LowerCaseInputStream extends FilterInputStream {

  public LowerCaseInputStream(InputStream in) {
    super(in);
  }

  public int read() throws IOException {
    int c = super.read(); return (c == -1 ? c : Character.toLowerCase((char)c));
  }

  public int read(byte[] b, int offset, int len) throws IOException {

    int result = super.read(b, offset, len);

    for (int i = offset; i < offset+result; i++) {
       b[i] = (byte)Character.toLowerCase((char)b[i]);
    }
    return result;
}
```

Padrões de Projeto

Solução do Exercício

```
public class InputTest {
  public static void main(String[] args) throws IOException {
    int c;

    try {
      InputStream in = new LowerCaseInputStream(
      new BufferedInputStream( new FileInputStream("test.txt")));

      while((c = in.read()) >= 0) {
         System.out.print((char)c);
      }

      in.close();

    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Padrões de Projeto

Exercício IV

Escreva uma classe que implemente um objeto que descreva um e-mail. Um e-mail tem um conteúdo na forma de uma cadeia de caracteres. Suponha que o e-mail tenha que possuir novas características: primeiro, o e-mail deve ter um descritor da companhia adicionado ao final do conteúdo original; segundo, o e-mail deve permitir o uso de mensagens criptografadas. Proponha uma solução em que os diferentes tipos de e-mail possam ser utilizados em um sistema cliente.

Referência para o exercício anterior:
http://java.dzone.com/articles/design-patterns-decorator

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books
- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman
- http://java.dzone.com/articles/design-patterns-decorator. Último acesso em 05/02/2013 - 20:49