Padrões de Arquitetura

Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

Introdução

- Temos vários padrões para arquitetar um software
- Para escolher qual padrão mais se encaixa no seu problema temos que
  - Realizar a declaração do problema
  - Verificar qual padrão tem um escopo apropriado para o problema
  - O padrão realmente resolve o problema declarado
  - O contexto do padrão corresponde ao contexto do problema
  - O padrão contém informações suficientes para você implementá-lo.
  - O padrão se ajusta ao que você já projetou

Introdução

- Saber o básico OO não o torna um bom designer OO.
- Bons designs OO são reutilizáveis, extensíveis e de fácil manutenção.
- Os padrões mostram como construir sistemas com boas qualidades de design OO.
- Os padrões são uma experiência comprovada em orientação a objetos.
- Os padrões não fornecem código, eles fornecem soluções gerais para problemas de design.
- Você os aplica de forma personalizada ao mesmo tipo de objeto.
- Os padrões não são inventados, eles são descobertos.

Introdução

- A maioria dos padrões e princípios aborda questões de mudança no software.
- A maioria dos padrões permite que alguma parte de um sistema varie independentemente de todas as outras partes.
- Freqüentemente, tentamos pegar o que varia em um sistema e encapsula-lo.
- Os padrões fornecem uma linguagem compartilhada que pode maximizar o valor de sua comunicação com outros desenvolvedores.

Padrões GoF - Classificação por Propósito

- Padrão Estratégia (Comportamental)
- Padrão Observador (Comportamental)
- Padrão Fachada (Estrutural)
- Padrão Decorador (Estrutural)
- Padrão Singleton (Criacional)

		Propósito		
		Criação	**Estrutura**	**Comportamento**
Escopo	**Classe**	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Builder Abstract Factory Prototype Singlenton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Classificação segundo GoF

Padrão Estratégia

Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

O que é o Padrão Estratégia?

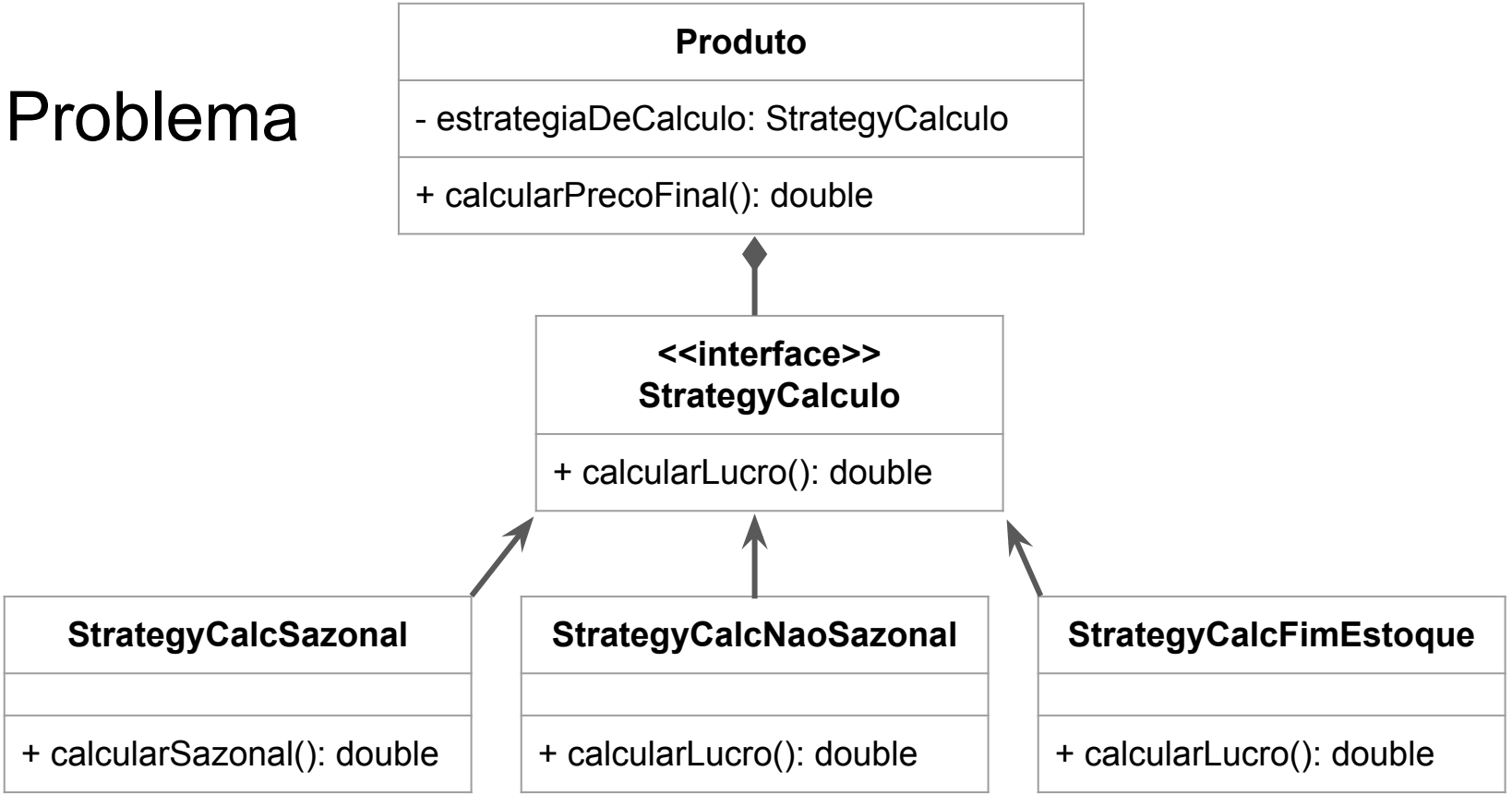
- O Padrão Estratégia delega as responsabilidades adquiridas pelas entidades, atribuindo, portanto, o comportamento.
- Comunicação entre os objetos é aprimorada, pois **há a distribuição das responsabilidades.**
- **O padrão Strategy permite definir novas operações sem alterar as classes dos elementos sobre os quais opera**.
- "Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. **Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam**." GOF(Gang of Four)

Problema

Problema

- A loja Passo Gourmet está tendo **problemas na oferta de preços de forma automática**
- Em diferentes épocas do ano um produto pode proporcionar um lucro maior ou menor para os comerciantes
- **Sazonal é sinônimo de estacional**, ou referente a uma estação.
- Por definição, o termo sazonal significa algo temporário, que possui uma época específica. Um produto sazonal é o oposto de um mercado perene, ou permanente.
- O preço dos produtos devem **mudar automaticamente de acordo com a época do ano e também se o produto está com poucas unidades em estoque**

Problema


Produto
- estrategiaDeCalculo: StrategyCalculo
+ calcularPrecoFinal(): double
<<interface>>
StrategyCalculo
+ calcularLucro(): double
StrategyCalcSazonal
+ calcularSazonal(): double
StrategyCalcNaoSazonal
+ calcularLucro(): double
StrategyCalcFimEstoque
+ calcularLucro(): double

Forma sem utilizar o Padrão Estratégia

Utilizando o Padrão Estratégia

Desafio

Desafio

- Utilize o método do Produto trocarDeEstrategia para verificar se o dia atual é em uma época sazonal ou não.
- Se o estoque tiver menos de 5 produtos, então a estratégia de cálculo será StrategyCalcFimEstoque
- Considere somente os meses sazonais mais importantes para o comércio listados abaixo:

mês 04 - Páscoa

mês 05 - Dia das Mães

mês 06 - Dia dos Namorados

mês 08 - Dia dos Pais

mês 10 - Dia das Crianças

mês 12 - Natal

Vantagens de Utilizar o Padrão Estratégia

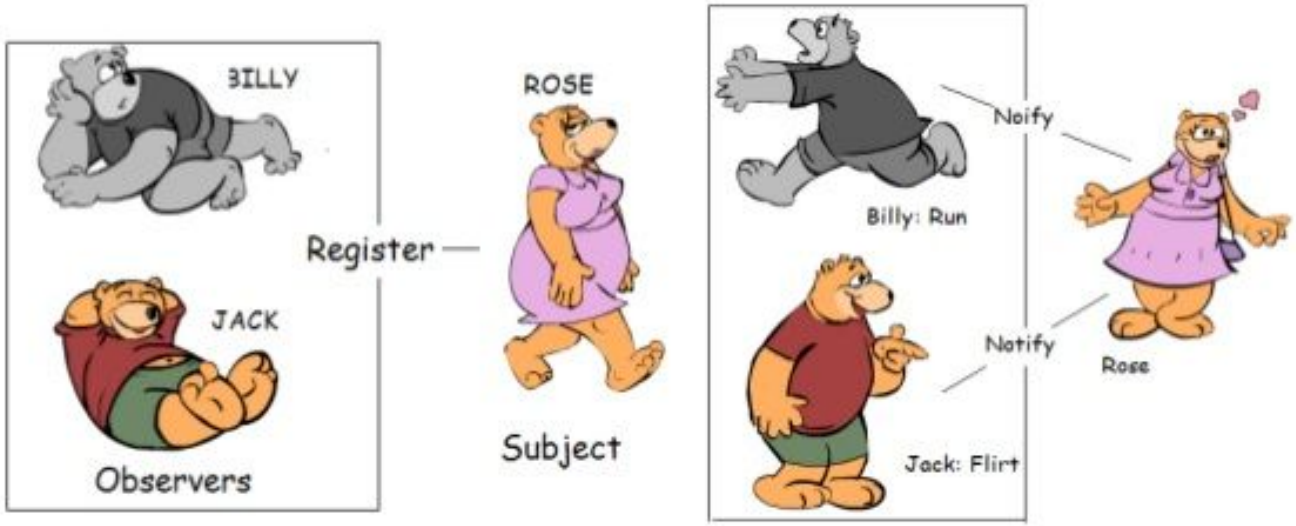
Vantagens de Utilizar o Padrão Estratégia

- **Família de Algoritmo**
- **Subclasses Alternativas**
- **Classes Estratégicas**
- **Escolha de implementações**
- **Clientes devem conhecer as classes Strategy**
- **Custo entre a comunicação Strategy e Context**
- **Maior número de objetos**

Padrão Observador

Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

O que é o Padrão Observador?



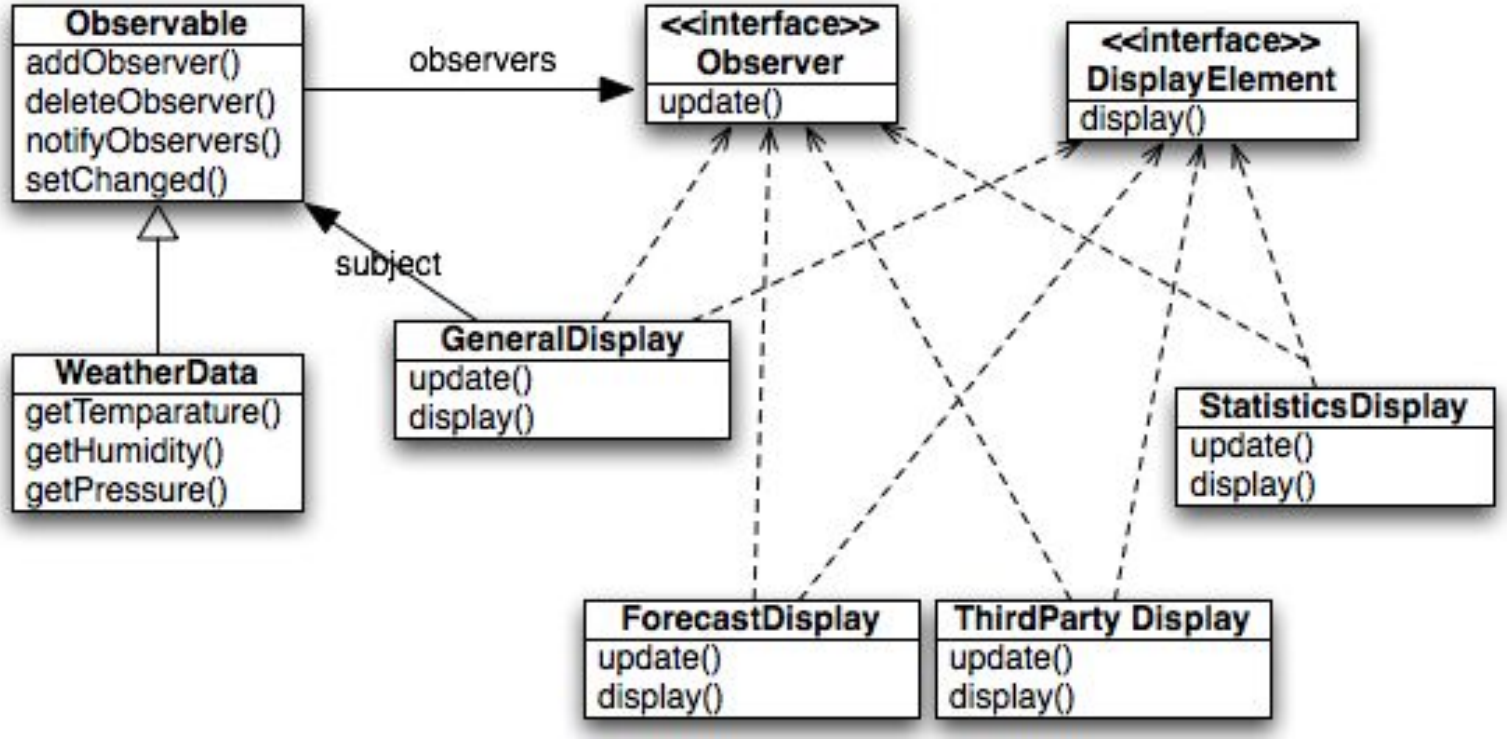
Observadores registram o Sujeito e aguardam notificação

Sujeito notifica Observadores sobre uma troca de estado. Observadores tomam uma ação

O **Padrão Observador** define uma dependência um-para-muitos entre os objetos para que quando um objeto muda de estado, **todos os seus dependentes são notificados e atualizados automaticamente**.

Problema

Problema


Observable
addObserver()
deleteObserver()
notifyObservers()
setChanged()
observers
<<interface>>
Observer
update()
<<interface>>
DisplayElement
display()
subject
WeatherData
getTemparature()
getHumidity()
getPressure()
GeneralDisplay
update()
display()
StatisticsDisplay
update()
display()
ForecastDisplay
update()
display()
ThirdParty Display
update()
display()

Forma sem utilizar o Padrão Observador

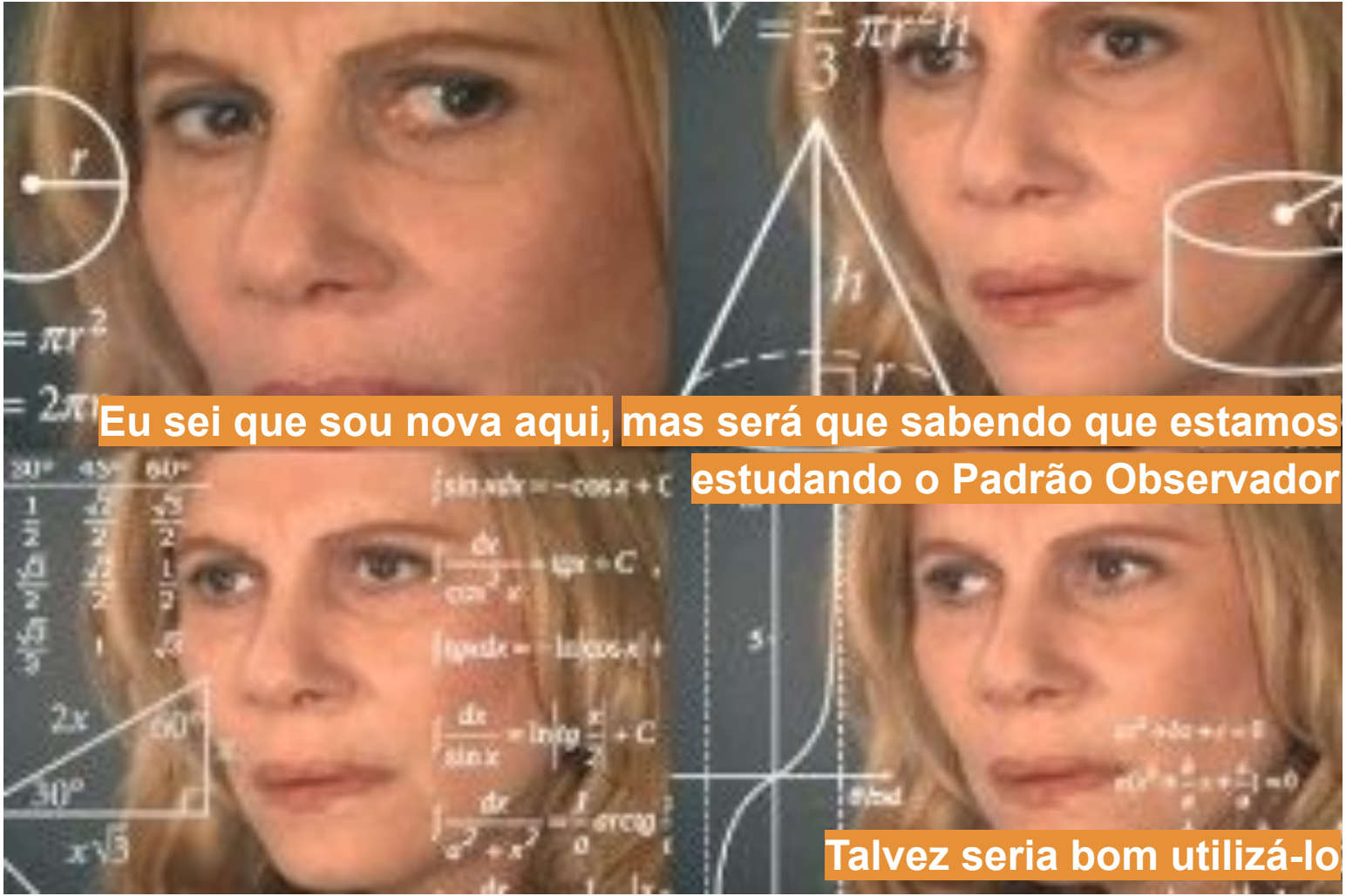
Forma sem utilizar o Padrão Observador

```
public void measurementsChanges(){
    float temp = getTemperature();
    float humity = getHumidity();
    float pressure = getPressure();
    currentContidionsDisplay.update(temp, humidity,
pressure);
    staticsDisplay.update(temp, humidity, pressure);
    forecastDisplay.update(temp, humidity, pressure);
}
```

Falta encapsulamento

Codificando em implementações concretas **não conseguiremos adicionar ou remover outros displays** sem fazer alterações no programa

Pelo menos temos uma classe comum chamada update()


Eu sei que sou nova aqui, mas será que sabendo que estamos
estudando o Padrão Observador
Talvez seria bom utilizá-lo

Utilizando o Padrão Observador

Desafio

Desafio

- Crie mais dois observadores **StaticsDisplay** e **ForecastDisplay** em que
  - **StaticsDisplay** informará a cada atualização quais as 3 ultimas temperaturas do Sujeito
  - **ForecastDisplay** enviará uma mensagem personalizada em relação a temperatura e a humidade:

a) Se a temperatura estiver menor que 10 graus celsius será uma mensagem de inverno, se entre 10 e 25 meia estação, maior que 25 graus celcius verão
b) Se a humidade estiver menor que 30% não chove, entre 30% e 70% talvez chova, maior que 70% então chove.

Desafio

```
Temperatura atual: 80.0F e 65.0% de humidade
Avg/Max/Min temperatura = 80.0/80.0/80.0
Recomendações: Hoje eh meia estacao, manga curta e talvez chova

Temperatura atual: 82.0F e 85.0% de humidade
Avg/Max/Min temperatura = 80.0/80.0/80.0
Recomendações: Hoje eh meia estacao, manga curta e leve o guarda
chuva

Temperatura atual: 80.0F e 49.0% de humidade
Avg/Max/Min temperatura = 80.0/80.0/80.0
Recomendações: Hoje eh meia estacao, manga curta e sem chuva
```

Vantagens de Utilizar o Padrão Observador

Vantagens de Utilizar o Padrão Observador

- O Padrão Observador define um relacionamento **um-para-muitos** entre os objetos.
- Atualizam Observadores usando uma **interface comum**.
- Os observadores são fracamente acoplados no sentido de que **o Sujeito não sabe nada sobre eles**, diferente de que eles implementam a interface Observer.
- **Não** é preciso uma **ordem específica** de notificação para observadores.
- Java tem várias implementações do Padrão Observer, incluindo o padrão **java.util.Observable**.
- Pode-se criar uma **própria implementação** Observador, se necessário.

Vantagens de Utilizar o Padrão Observador

- O Swing faz uso intenso do Observer Pattern, assim como muitos frameworks GUI.
- Você também **encontrará o padrão em muitos outros lugares**, incluindo JavaBeans e RMI.

Padrão Decorador

Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

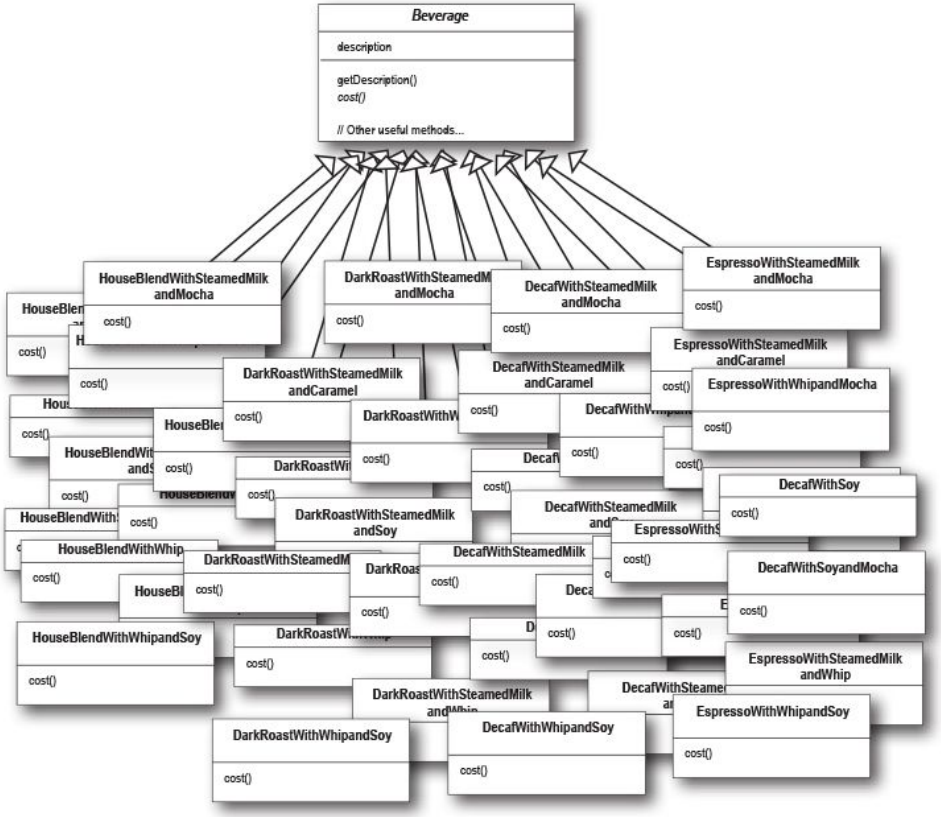
O **Padrão Decorador** atribui responsabilidades adicionais a um objeto **dinamicamente**.
Os decoradores fornecem uma alternativa flexível à criação de subclasses para estender a funcionalidade.

Problema


water
espresso
AMERICANO
milk foam
milk
espresso
LATTE
milk foam
half and half
milk and cream
espresso
COFFEE BREVE
milk
espresso
CAFE AU LAIT
whipped cream
espresso
VIENNA COFFEE
whipped cream
whiskey
espresso
IRISH COFFEE
milk foam
chocolate
milk
espresso
CAPPUCCINO
espresso
ESPRESSO
milk foam
espresso
FLAT WHITE
ice cream
grated chocolate
espresso
GLACE
brandy
espresso
CORRETTO
espresso
LUNGO
milk foam
espresso
MACCHIATO
lemon
espresso
ESPRESSO ROMANO
chocolate
whipped cream
ice cream
espresso
ICED COFFEE
cinnamon
whipped cream
espresso
ESPRESSO CON PANNA
milk foam
chocolate syrop
espresso
chocolate
MAROCHINO
milk
ice
espresso
FRAPPE
vanilla sugar
cream
espresso
RAF
whipped cream
milk
espresso
chocolate
BICERIN
cream
espresso
DEMI-CREME
milk foam
espresso
GALAO
espresso
RISTRETTO
milk
caramel
espresso
FREDDO
milk foam
milk
espresso
chocolate syrop
MOCHA

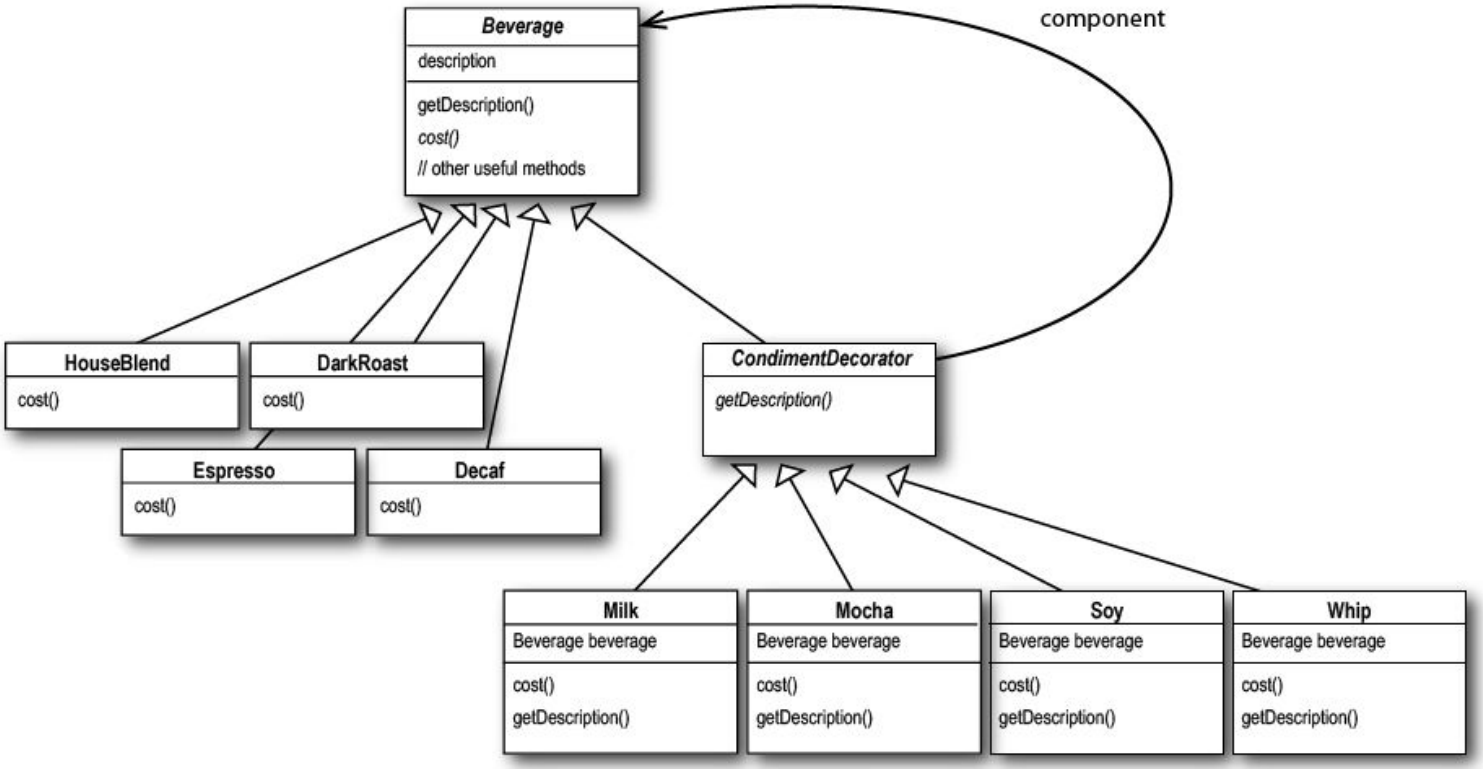
Forma sem utilizar o Padrão Decorador

Forma sem utilizar o Padrão Decorador



Utilizando o Padrão Decorador

Utilizando o Padrão Decorador



Desafio

Desafio

- Crie mais opções de sabores para a cafeteria com a bebida DarkRoast() e condimento extra Whip() de modo que ao descomentar as linha no main os preços e descrições possam ser atualizados ao chamar essas duas classes.

```
Espresso $1.99
House Blend Coffee, Soy, Mocha, Whip $1.59
Dark Roast Coffee, Mocha, Mocha, Whip $1.49
```

Vantagens de Utilizar o Padrão Decorador

Vantagens de Utilizar o Padrão Decorador

- **Herança é uma forma de extensão**, mas **não necessariamente a melhor** maneira de alcançar flexibilidade em nossos projetos.
- Em nossos projetos, devemos permitir que o comportamento seja **estendido sem a necessidade de modificar código existente**.
- A **composição e a delegação** geralmente podem ser usadas para **adicionar** novos comportamentos em **tempo de execução**.
- **O Padrão Decorador fornece uma alternativa à subclasse para estender o comportamento.**
- O Padrão Decorador envolve um conjunto de classes decorator que são usadas para embrulhar componentes de concreto.

Vantagens de Utilizar o Padrão Decorador

- Decoradores mudam o comportamento de seus componentes **adicionando novas funcionalidades** antes e / ou depois (ou mesmo no lugar de) chamadas de método para o componente.
- Você pode envolver um componente com **múltiplos número de decoradores**.
- Decoradores podem resultar em **muitos objetos em nosso design, e o uso excessivo pode ser complexo**.

Padrão Fachada

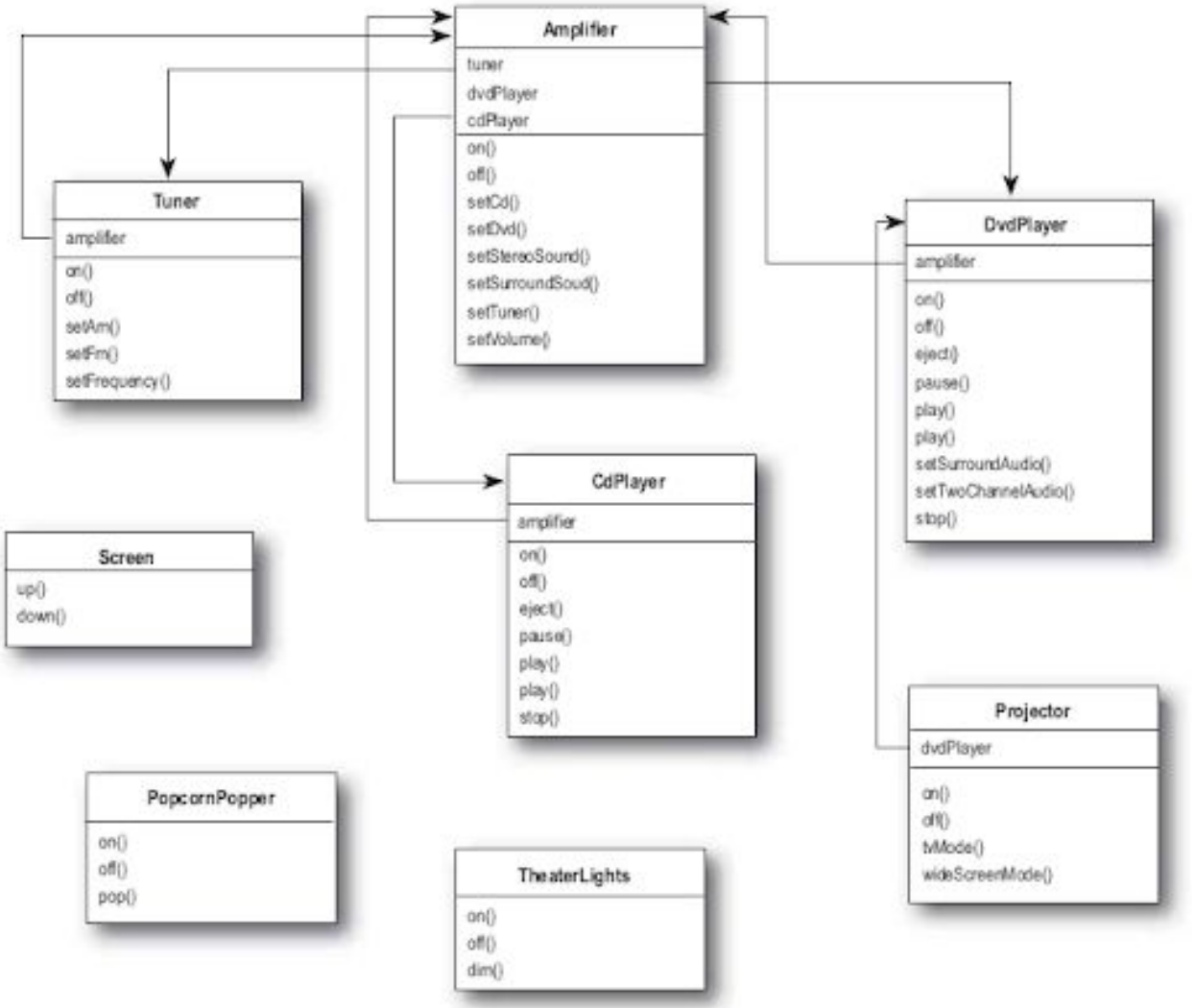
Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

O **Padrão Fachada** fornece uma interface unificada para um conjunto de interfaces em um subsistema. Fachada define uma interface de nível superior que torna o subsistema mais fácil de usar

Problema


Cinema Paradiso
A director returns home after 30 years, reminiscing about his first love and the man who introduced him to film.

Problema


Amplifier
tuner
dvdPlayer
cdPlayer
on()
off()
setCd()
setDvd()
setStereoSound()
setSurroundSoud()
setTuner()
setVolume()
Tuner
amplifier
on()
off()
setAm()
setFm()
setFrequency()
DvdPlayer
amplifier
on()
off()
eject()
pause()
play()
play()
setSurroundAudio()
setTwoChannelAudio()
stop()
CdPlayer
amplifier
on()
off()
eject()
pause()
play()
play()
stop()
Screen
up()
down()
Projector
dvdPlayer
on()
off()
tvMode()
wideScreenMode()
PopcornPopper
on()
off()
pop()
TheaterLights
on()
off()
dim()

Forma sem utilizar o Padrão Fachada

Forma sem utilizar o Padrão Fachada

```
public class HomeTheaterTestDrive{
    public static void main(String[] args){
      ...
        popper.on();
        popper.pop();
        lights.dim(10);
        screen.down();
        projector.on();
        projector.wideScreenMode();
        amp.on();
        amp.setDvd(dvd);
        amp.setSurroundSound();
        amp.setVolume(5);
        dvd.on();
        dvd.play(movie);
      ...
    }
}
```

Utilizando o Padrão Fachada

Desafio

Desafio

- Descomente os códigos comentados e **implemente cada uma das classes presentes** assim como seus métodos e atributos
- Ajuste **HomeTheareFacade** para receber todas as classes
- A resposta final deve ser conforme a do slide

Desafio

```
Prepare-se para assistir a um filme ...
Pipoqueira Pipocando
Pipoca Pipocada na Pipoqueira!
Luzes do Cinema a 10%
Tela do Cimema para baixo
Amplificador Top-O-Line ligado
O amplificador Top-O-Line configurou o reprodutor de DVD para o
reprodutor de DVD Top-O-Line
Som surround do amplificador Top-O-Line ligado (5 alto-falantes, 1
subwoofer)
Volume do amplificador Top-O-Line para 5
DVD Player Top-O-Line ligado
Leitor de DVD Top-O-Line reproduzindo MARVEL Os Vingadores
```

Desafio

```
Fechando o cinema …
Pipoqueira para Pipoca desligada
Luzes do Cinema ligadas
Tela do Cinema para cima
Amplificador Top-O-Line desligado
O DVD Player Top-O-Line parou MARVEL Os Vingadores
Finalizacao do reprodutor de DVD Top-O-Line
DVD Player Top-O-Line desligado
```

Vantagens de Utilizar o Padrão Fachada

Vantagens de Utilizar o Padrão Fachada

- Quando você precisa **simplificar e unificar uma grande interface ou conjunto complexo de interfaces**, use uma fachada.
- Uma fachada **desacopla um cliente de um subsistema complexo**.
- Implementar uma fachada requer que **compunham a fachada com seu subsistema** e usar delegação para executar o trabalho de fachada.
- **Você pode implementar mais de uma fachada para um subsistema.**
- Uma fachada "envolve" um conjunto de objetos para simplificar.

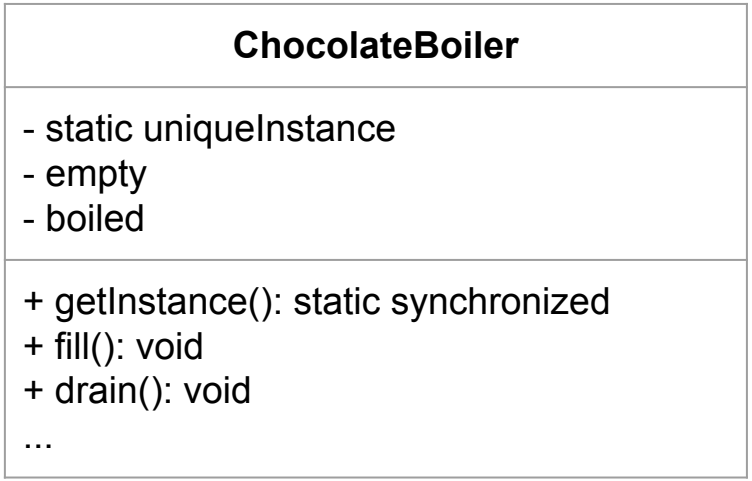
Padrão Singleton

Aluna: Sthefanie Passo
Professor: Moisés Carvalho
Projeto e Arquitetura de Software

O **Padrão Singleton** garante que uma classe tenha somente uma instância e fornece um ponto global de acesso a ela

Problema

Problema


ChocolateBoiler
- static uniqueInstance
- empty
- boiled
+ getInstance(): static synchronized
+ fill(): void
+ drain(): void
...

Forma sem utilizar o Padrão Singleton

Forma sem utilizar o Padrão Singleton

```
public class Industry{
    public static void main(String args[]){
        ChocolateBoiler florybal = new ChocolateBoiler();
        florybal.ChocolateBoilerStatus("Florybal");
        florybal.fill();

        ChocolateBoiler gramadoWay = new ChocolateBoiler();
        gramadoWay.ChocolateBoilerStatus("Gramado Way");
        gramadoWay.fill();
        gramadoWay.ChocolateBoilerStatus("Gramado Way");
    }
}
```

Utilizando o Padrão Singleton

Desafio

Desafio

- Ajude mais uma loja Caracol Chocolates a ser automatizada com essa aplicação.
- Como cada loja representa uma thread então faça dupla verificação no getIstance() de ChocolateBoiler e acrescente uma sequência de métodos para o Caracol em que o ultimo print deve ser

```
Industry: Caracol
Empty: true
Boil: false
********************
```

Vantagens de Utilizar o Padrão Singleton

Vantagens de Utilizar o Padrão Singleton

- O Singleton Pattern garante que você tenha no máximo **uma instância de uma classe em seu aplicativo**.
- O Singleton Pattern também fornece um **ponto de acesso global** para essa instância.
- A implementação do Java do Singleton Pattern faz uso de um **construtor privado**, um método estático combinado com uma **variável estática**.
- Examine seu desempenho e restrições de recursos e escolha cuidadosamente uma implementação Singleton apropriada **para aplicativos multithread** (e devemos considere todos os aplicativos multithread!).

Vantagens de Utilizar o Padrão Singleton

- **Cuidado com a implementação de bloqueio verificado duas vezes**; não é thread-safe em versões anteriores ao Java 2, versão 5.
- Tenha **cuidado se estiver usando vários carregadores de classes**; isso poderia **prejudicar** o Singleton e resultar em várias instâncias.

Leitura base

1. Head First Design Pattern - Freeman

SOLI DEO GLORIA