INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS
*CAMPUS MONTES CLAROS* -MG
CIÊNCIA DA COMPUTAÇÃO

Rafael José Braga Coelho

# Bridge Pattern

Documento PDF entregue ao Professor Luis Antonio Guisso Lopes como parte das exigências de avaliação da disciplina de Programação Orientada a Objetos de Bacharelado em Ciência da Computação do Instituto Federal do Norte de Minas Gerais, Campus Montes Claros.

MONTES CLAROS - MG

2024

# 1.Introdução

## 1.1Comprometimento:

Este relatório apresenta uma breve explicação dos padrões de projeto no desenvolvimento de software e um estudo detalhado sobre os padrões de projeto Abstract Factory e bridge juntamente aos diagramas de classes implementados junto ao seminário.

## 1.2Compreensão básica e motivação:

Podemos interpretar os padrões de projeto como um cinto de ferramentas, com cada uma das ferramentas sendo capazes de solucionar vários problemas que são recorrentes no desenvolvimento de software, onde por serem padrões podem ser facilmente uma forma de ter sua ideia de projeto repassada a outros programadores e garantir uma melhor coordenação e fluxo de trabalho, seus 3 tipos são:

Criacional: Foca na criação de objetos, melhorando na reutilização de código já existente

Estrutural: Foca no uso destes objetos pensando em uma estrutura maior que flexibiliza e faz um uso mais eficiente destes objetos.

Comportamental: Gera uma comunicação e uma entrega de tarefas mais objetiva entre os objetos

# 2 Entendendo o Abstract factory

Nesse caso vamos entender a utilização deste padrão de projetos que promove o desacoplamento do usuário da implementação aproveitando da flexibilidade e extensibilidade da nossa implementação, tornando mais simples o processo de adicionar novas famílias de produtos sem que haja modificações do código já existente.

# 2.1 Padrão Abstract Factory

Este padrão de projeto definido como um padrão Criacional que tem por nome Abstract Factory onde ela “abstrai” para o usuário a “fabricação” das instâncias, a partir de 4 tipos de classes conceitos:

### 2.1.1 Produtos abstratos:

Diga que os produtos(classes abstratas) abstratos seriam a ideia base do produto de onde os outros vão pegar os pontos principais e implementar de acordo as suas necessidades específicas.

### 2.1.2 Produtos concretos:

Os produtos concretos(classes que generalizam dos produtos abstratos, que por sua vez são classes abstratas) são quando a ideia saiu do papel e é levado de fato para o mercado, trazendo todas as características do produto abstrato e implementando suas peculiaridades.

### 2.1.3 Fábricas Abstratas:

As fábricas abstratas são o que dá nome ao pattern é a principal ideia por trás de todo este padrão de projeto, ela é quem vai definir as fábricas concretas quais tipos de produtos(instâncias) serão produzidas(instanciadas) por todas as fábricas concretas relacionadas ao tópico.

### 2.1.4 Fábricas Concretas:

As fábricas concretas são o que realmente é utilizado no código pelo usuário, são elas que vão instanciando os produtos concretos necessários para a implementação, assim fechando nossa linha de produção com instâncias especializadas para cada uso descrito dos produtos concretos.

### 2.1.5 Utilização na vida real:

Interfaces gráficas precisam repetir o seu mesmo comportamento por várias plataformas, plataformas essas que têm formas diferentes de lidar com programas, se o funcionamento padrão da interface for feito conseguimos a partir deste padrão de projeto fazê- lo funcionar da mesma forma apenas adaptando o mesmo produto abstrato para outro tipo de produto concreto.

## 2.2 Prós e Contras

A vantagem deste padrão de projeto é a sua alta independência e a abstração para o cliente de cada produto concreto onde podemos implementar outros produtos concretos do mesmo tipo sem afetar outros produtos concretos ao mesmo tempo o cliente não precisa estar atento à implementação de forma detalhada sobre cada produto.

O problema se dá onde existe a introdução de um novo produto abstrato, onde a estrutura que extende desde o começo da implementação até o final da “linha de produção” vai sofrer alterações de implementação

## 2.3. Aplicação

### 2.3.1 Definindo o problema

O problema utilizado para demonstrar a capacidade deste padrão de fábrica de simplificar a instalação de objetos(criação de produtos concretos) será sobre fábricas de veículos,estas fábricas criam carros e motos que podem ser do tipo flex ou elétricas.

### 2.3.2 Solução

A solução deste problema na minha visão pode ser separada a partir dos mesmos tópicos de antes na definição onde teremos:

### 2.3.3 Produtos abstratos:

Carro e Moto são classes abstratas ou interfaces que definem o tipo de veículos que podem ser criados pelas fábricas, no meu caso são só abstratas mesmo .

### 2.3.4 Produtos concretos:

- **Veículos Flex:**

CarroFlex e MotoFlex são subclasses ou implementações das classes abstratas Carro e Moto, respectivamente. Eles representam veículos que funcionam com combustível Flex.

- **Veículos Elétricos**:

CarroEletrico e MotoEletrica são subclasses ou implementações das classes abstratas Carro e Moto, representando veículos elétricos.

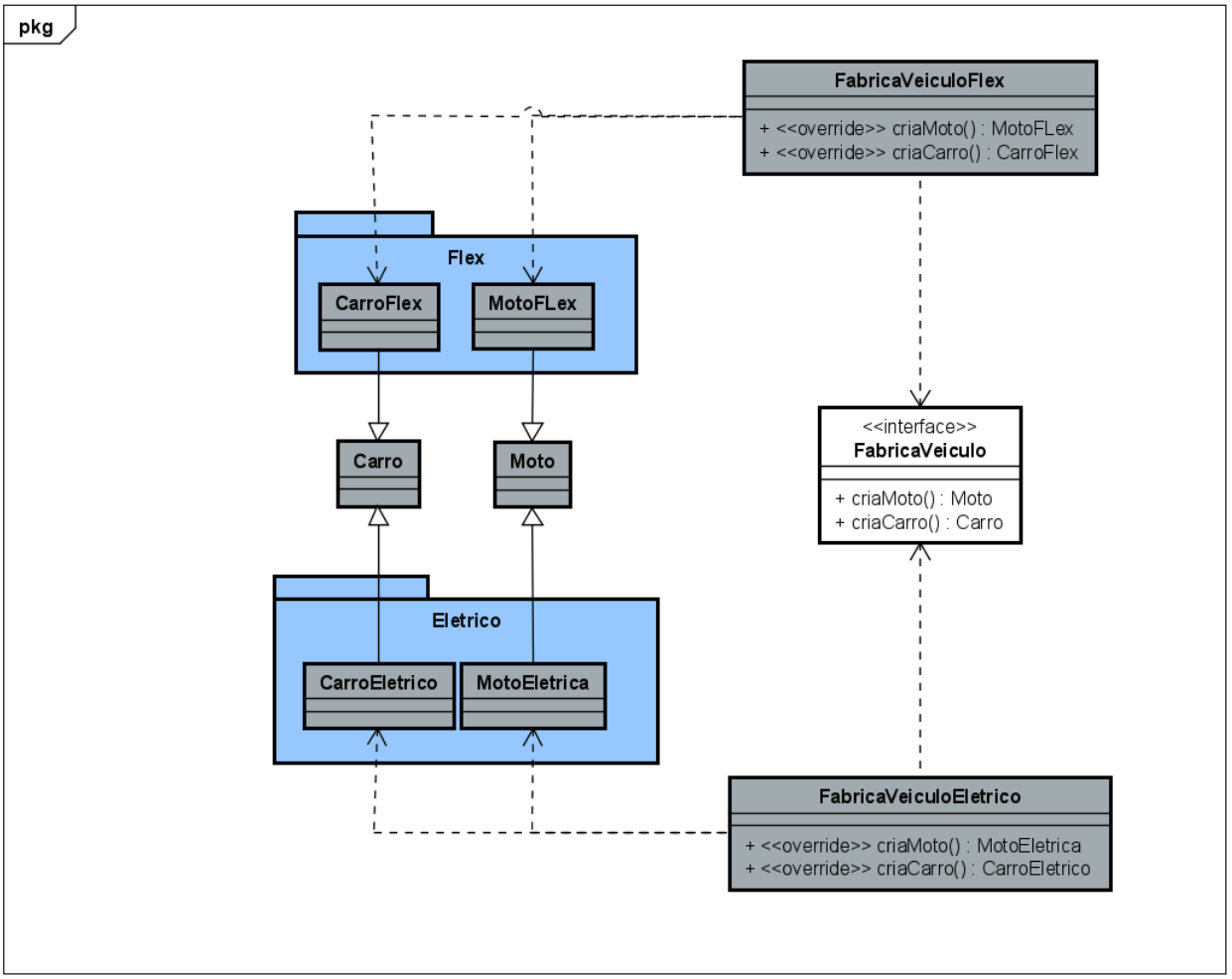
### 2.3.5 Fábricas Abstratas:

A interface das fábricas concretas chamada de FabricaVeiculo que define as implementações das fábricas concretas a partir da ideia base de instanciar as classes Carro e Moto.

### 2.3.6 Fábricas Concretas:

Suas funções de criação são herdadas das fábricas abstratas implementando as suas criações especificadas onde a FábricaVeiculoFlex vai instanciar CarroFlex,MotoFlex e a FabricaVeículoEletrico que vai instanciar CarroEletrico e MotoEletrica.

## 2.4. Diagrama




## 2.5 Conclusão

O padrão de projeto Abstract Factory é uma boa ferramenta de design de software que promove a flexibilidade e a reutilização de código. A partir de sua grande abstração e consistência nas suas generalizações, ele garante que os componentes do sistema sejam compatíveis e intercambiáveis. Este padrão é especialmente útil em sistemas que precisam ser independentes da forma como seus objetos são criados e feitos para trabalhar com diferentes famílias de produtos.

# 3.Padrão Bridge

nesse caso vamos entender a utilização deste padrão de projetos que promove o desacoplamento entre abstrações e implementações. O objetivo principal deste relatório é explorar o conceito, os benefícios e as motivações para sua utilização em cenários onde múltiplas variações de abstração e implementação são necessárias.

Este padrão de projeto definido como um padrão estrutural que se chama bridge por justamente haver uma “ponte” entre duas hierarquias, a abstração e implementação(Interface) que podem ser desenvolvidas de forma independente , essas hierarquias através desta ponte facilitam a interação com as classes que estão do outro lado da ponte, que ao contrário de onde fariam implementações de classes específicas para cada situação,
vamos ter essas várias situações descritas na abstração e as classes que vão poder interagir com essas situações na implementação.

O uso deste padrão de projeto parece vir a calhar quando se trata de renderizações na implementação do sistema operacional, já que estes estão a toda hora interagindo com coisas novas e dividindo estas novidades a serem implementadas que vem para tanto a implementação e a abstração.

## 3.1. Aplicação

### 3.1.1 Definindo o problema

Definido o padrão, vamos entender melhor sobre como a sua aplicação em um projeto sobre mandar várias formas de mensagens a partir de vários meios pode ajudar na compreensão de um desenvolvimento ao qual o bridge pattern ajuda na resolução de problemas do tipo.

De forma simples, se trata de enviar mensagens que precisam ser modificadas ou não de várias formas a partir de vários serviços diferentes, neste caso encontramos um problema que pode ser implementado a partir do padrão estudado neste relatório.

No meu exemplo entende-se que vamos enviar mensagens a partir dos serviços de SMS e Email, a questão é que os usuários querem escolher se sua mensagem será criptografada ou não.

### 3.1.2 Solução explicativa(sem bridge)

A ideia que o padrão bridge tenta correr, é de implementar cada um dos casos,
Onde fazemos várias classes para vários casos, aos quais enviar por SMS nos daria o trabalho de implementar as classes SmsCriptografado, SmsNormal, e classes de enviar por Email sendo estas EmailCriptografado e EmailNormal.

**percebe que o nosso problema escala:**
Caso haja 3 serviços de mensagem teríamos de fazer 6 classes implementadas,
e caso também fosse 3 formas de mensagem teria 9 classes a serem implementadas.

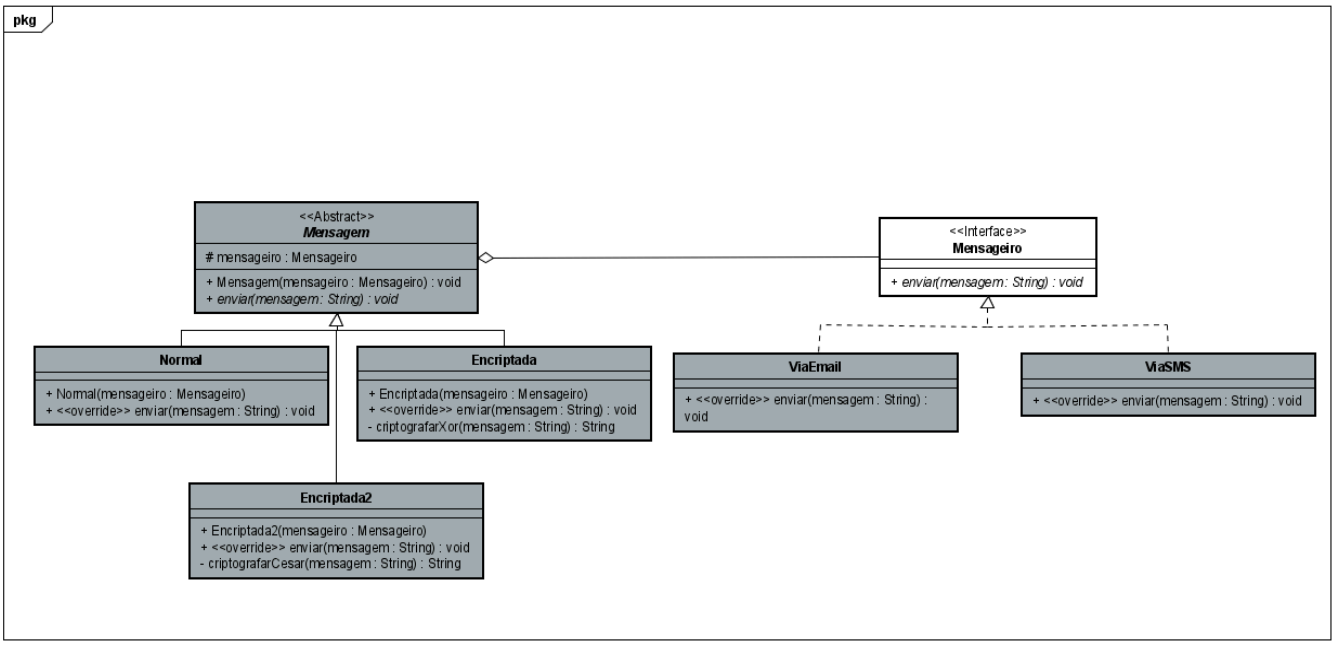
### 3.1.3 Solução Implementada com bridge

Para resolvermos esse problema com o padrão de projeto bridge, vamos separar a abstração da implementação, ou seja, vamos “implementar” o envio das mensagens de SMS e Email, enquanto na “abstração” teremos a maneira para definir qual forma a mensagem vai tomar no envio Criptografada e normal. (ou no exemplo implementado Encriptada,Encriptada2,Normal.)

**agora, se o nosso problema escalar:**

Caso haja 3 serviços de mensagem ao invés de 2 e 2 formas de mensagem ,vamos implementar 5 classes
sendo essas 2 de tipo mensagem e 3 de tipo serviços(“no exemplo mensageiro”),
e caso fosse 3 serviços de mensagem e 3 formas de mensagem ao invés de 2 teria 6 classes,
3 de serviço e 3 de mensagem.

## 3.2. Diagrama




## 3.3. Conclusão

O padrão de projeto bridge consegue implementar e abstrair de forma bem consistente um problema, onde se teria que ser refatorado várias vezes o mesmo código, em função de resolver o mesmo problema para várias situações diferentes, porém para um caso de uma classe já bem implementadas e mais coesa, pode ser que o código acabe mais complicado do que aquilo que já estava pronto.

## 4.Referências


REFACTORING GURU. *Bridge Design Pattern*. Disponível em: https://refactoring.guru/design-patterns/bridge. Acesso em: 13 set. 2024.
CODING WITH YALCO. *07. Bridge Pattern*. Vídeo. Disponível em: 07. Bridge Pattern (youtube.com). Acesso em: 13/09/2024
REFACTORING GURU. *Abstract Factory Design Pattern*. Disponível em: https://refactoring.guru/design-patterns/abstract-factory. Acesso em: 14 set. 2024.

BAELDUNG. *Abstract Factory Pattern in Java*. Disponível em: https://www.baeldung.com/java-abstract-factory-pattern. Acesso em: 14 set. 2024.

CÓDIGO SIMPLES. *Padrão de Projeto - Abstract Factory* [Vídeo]. 2020. Disponível em: https://www.youtube.com/watch?v=k-QhJDhxGc0. Acesso em: 14 set. 2024.

REFATORAÇÃO GURU. Por que aprender padrões de projeto? Disponível em: https://refactoring.guru/design-patterns/why-learn-patterns. Acesso em: 16 set. 2024.
REFACTORING GURU. *Bridge Design Pattern*. Disponível em: https://refactoring.guru/design-patterns/bridge. Acesso em: 13 set. 2024.