Abstract Factory Treino

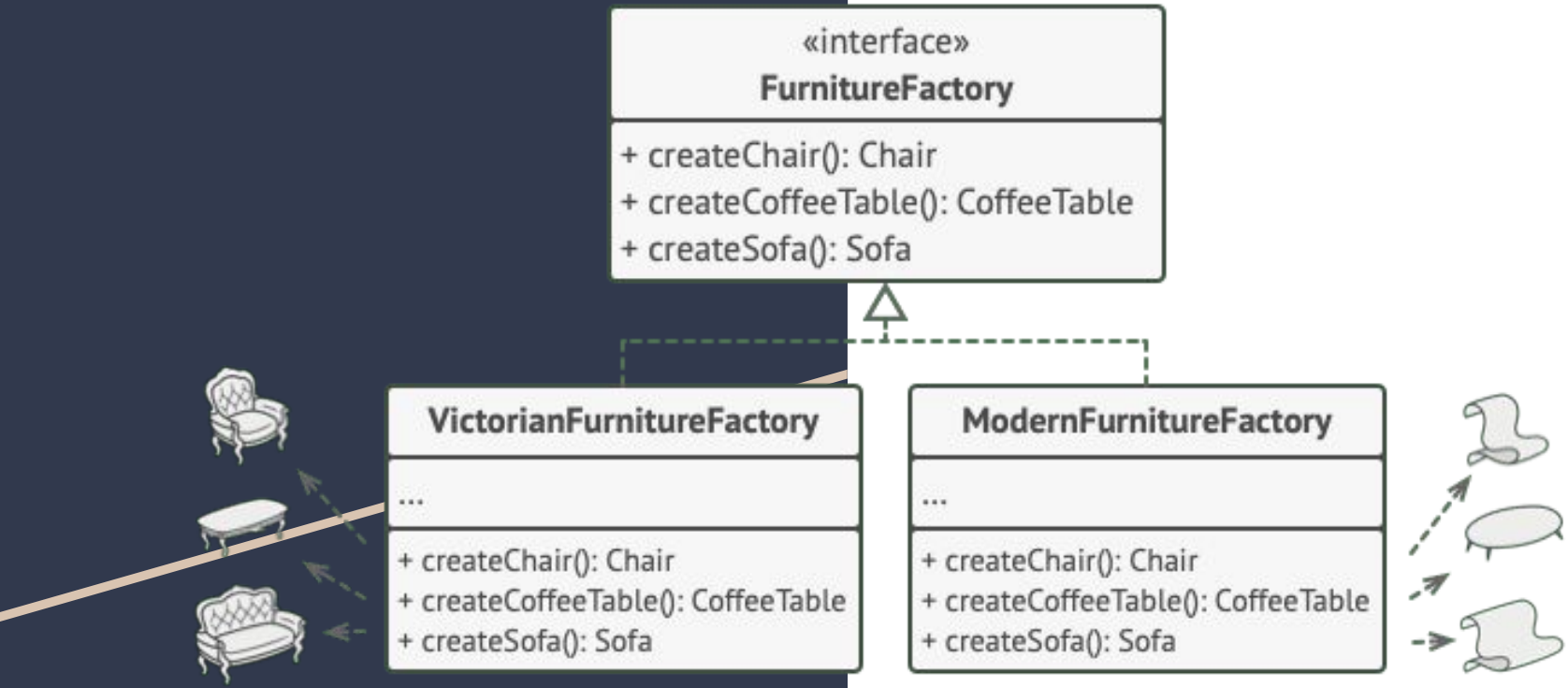
Abstract Factory

Ele faz parte dos **Patterns Criacionais**, os **padrões criacionais são** técnicas que ajudam controlar como as instâncias de classes **são** criadas

Objetivo / intenção: Permite a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe **concreta** seja especificada.

Motivação: O objetivo em empregar o padrão é isolar a criação de objetos de seu uso e criar famílias de objetos relacionados sem ter que depender de suas classes concretas. Isto permite novos tipos derivados de ser introduzidas sem qualquer alteração ao código que usa a classe base;

Estrutura


«interface»
FurnitureFactory
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
VictorianFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
ModernFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa

Problema

Imagine o seguinte exemplo:

Você está encarregado de desenvolver elementos visuais para interfaces gráficas, elementos como ScrollBars, Menus e etc.

Esses elementos visuais têm diferentes implementações para cada família de implementação gráfica, como o Microsoft Windows, o MAC, e Linux.

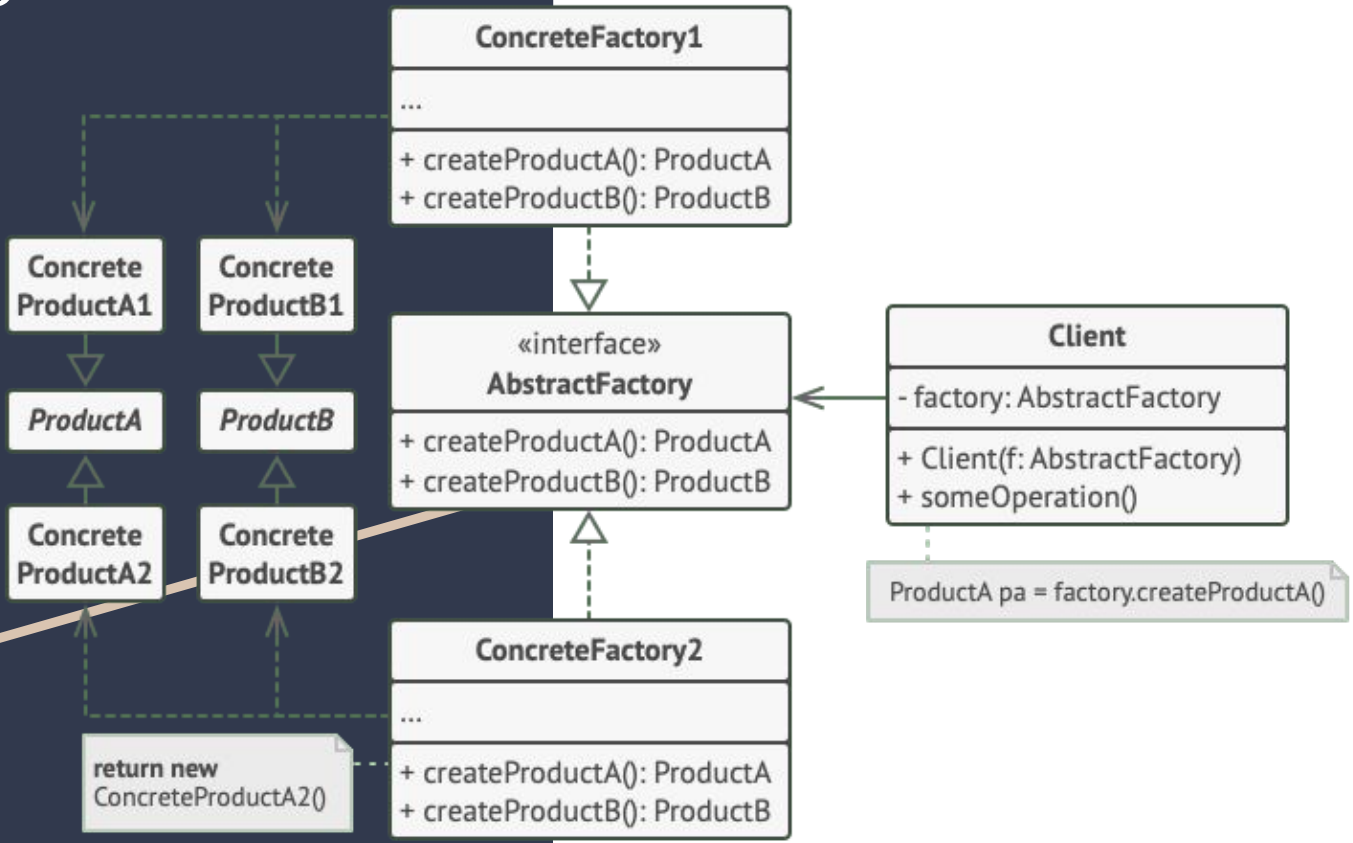
Solução

Nesse caso, a solução consiste em duas partes:

1. Crie **interfaces padrões** para os diferentes produtos dessa família (ScrollBar, Menu...). E todo o sistema vai trabalhar **apenas** com essas **interfaces** que você definiu.

2. Defina uma **Abstract Factory**, que tem os métodos de instanciação para cada uma dessas interfaces definidas acima.

Solução


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA()
ProductA
ProductB
Concrete ProductA2
Concrete ProductB2
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

Pontos Positivos

O ponto principal é que este pattern deixa seu sistema **independente** das diferentes famílias, ou seja, garante o **baixo acoplamento**.

Outro ponto positivo é que este pattern permite adicionar, remover ou modificar **rapidamente** qual família de produtos deseja-se usar.

Pontos Negativos

Já um ponto negativo desse pattern é que a adição ou remoção de um produto da família **exige a modificação da AbstractFactory**, o que causa um grande **overhead**, pois deve-se modificar **todas** as **implementações** da Factory e o cliente que usa a **AbstractFactory**.

Aplicabilidade

Cenários onde uma família de produtos ou classes precisa ser instanciado, sem dependência de suas classes concretas, como no exemplo citado, onde você tem elementos visuais (produtos), como Window, ScrollBar, Menu e etc.

Esses elementos visuais têm diferentes implementações para cada família de implementação gráfica, como o Microsoft Windows, o MAC, e X do Linux;

Dar o outro exemplo da pizzaria.

Referências

BIGARDI, G. B. **Arquitetura e desenvolvimento de software — Parte 2 — Abstract Factory**. Disponível em: <https://gbbigardi.medium.com/arquitetura-e-desenvolvimento-de-software-parte-2-abstract-factory-f603ccc6a1ea>. Acesso em: 5 abr. 2022.

GAMMA, E.; AL, E. **Design patterns : elements of reusable object-oriented software**. Boston: Addison-Wesley, 2016.

Abstract Factory. Disponível em: <https://refactoring.guru/pt-br/design-patterns/abstract-factory>.