Design Patterns (Padrões de Projetos)

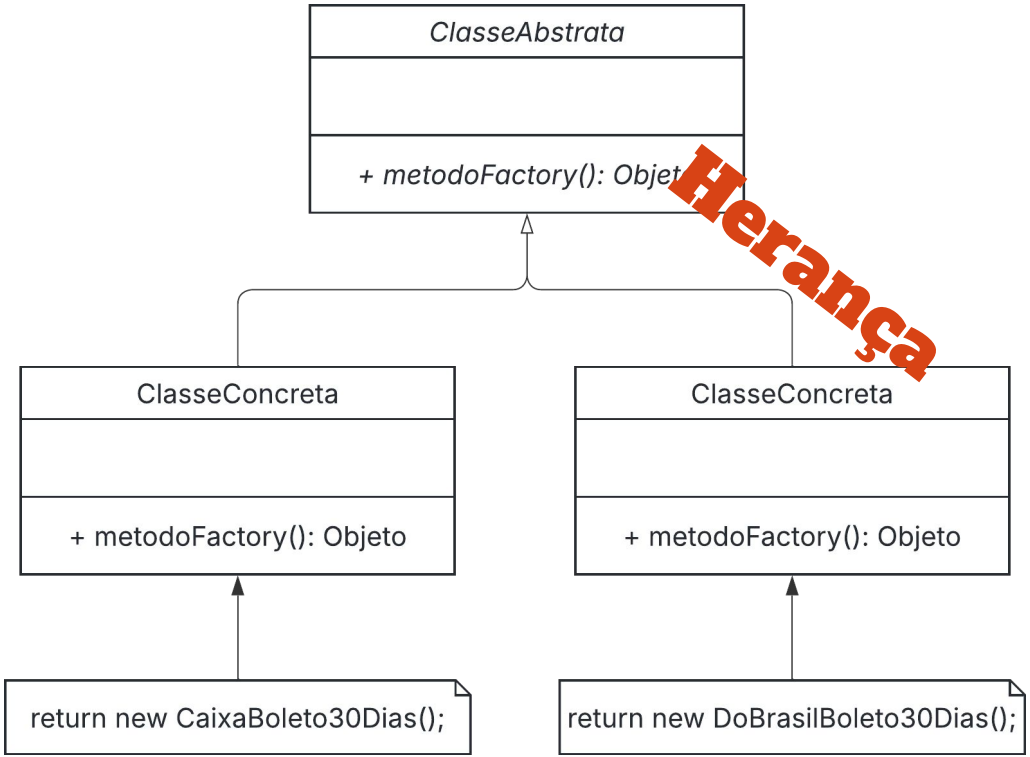
Abstract Factory

Abstract Factory

O **Abstract Factory** é um **padrão de projeto criacional** que fornece uma interface para criar **famílias de objetos relacionados** sem especificar suas classes concretas.

Diferente do **Factory Method**, que cria um único tipo de objeto, o **Abstract Factory** é usado quando precisamos garantir que múltiplos objetos criados sejam compatíveis entre si.

Factory Method

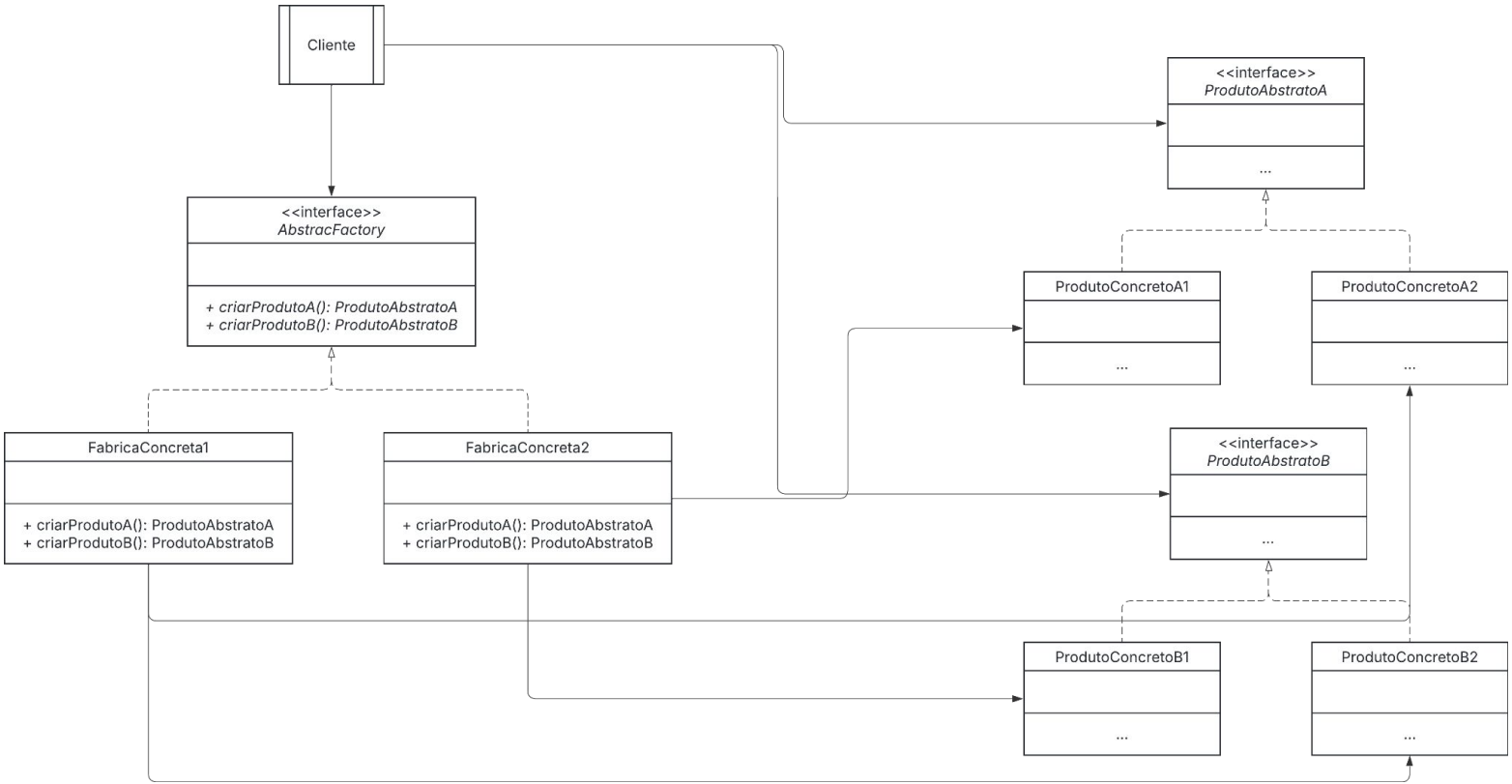


Problema

Imagine que estamos desenvolvendo uma biblioteca de componentes para uma aplicação que pode rodar em **Android e iOS**. Cada sistema tem sua própria aparência para botões e caixas de texto, então precisamos garantir que os componentes de um mesmo sistema operacional tenham um estilo consistente.

Se criarmos esses componentes diretamente no código, teremos que verificar qual sistema está sendo usado toda vez que criarmos um botão ou uma caixa de texto, resultando em **alto acoplamento e código difícil de manter**.

Abstract Factory



Quando usar?

- Quando um sistema deve ser independente de como seus produtos são criados, compostos ou representados.
- Quando um sistema deve ser configurado com uma dentre múltiplas famílias de produtos.
- Quando um familias de objetos relacionados foi projetada para ser usada em conjunto e é necessário impor essa restrição.
- Quando se deseja fornecer uma biblioteca de produtos e se deseja revelar para o cliente apenas suas interfaces, e não suas implementações.

Consequências

- Promove o isolamento de classes concretas.
- Facilita a troca de famílias de produtos.
- Promove a consistência entre produtos.
- Suportar novos tipos de produtos é difícil. (consequência negativa)
- Criar novos produtos de um tipo já existente é fácil.

Exercícios

Fabricação de brinquedos

Uma fábrica de brinquedos deseja criar diferentes **tipos de brinquedos** com base no **material** usado na fabricação. Existem dois materiais principais: **Plástico** e **Madeira**.

Cada brinquedo deve implementar um método play(), que exibe uma mensagem dizendo como ele é usado.

Tarefa

1. Implemente uma **Abstract Factory** que possa criar dois tipos de brinquedos: **Carrinho** e **Boneca**.
2. Crie duas fábricas concretas: **Fábrica de Plástico** e **Fábrica de Madeira**.
3. No código cliente, instancie cada fábrica e teste os brinquedos criados.

Exercícios

Criação de Criaturas Fantásticas

Um jogo de RPG precisa de um sistema para criar **criaturas mágicas** de diferentes reinos:

- **Reino do Fogo** → Criaturas como **Dragões e Salamandras**
- **Reino da Água** → Criaturas como **Serpentes Marinhas e Tritões**

Cada criatura deve ter um método attack(), que imprime uma mensagem personalizada de ataque.

Tarefa

1. Modele uma **Abstract Factory** para criar **criaturas mágicas**.
2. Crie fábricas concretas para os dois reinos (**Reino do Fogo** e **Reino da Água**).
3. No código cliente, instancie cada fábrica e faça as criaturas atacarem.

Exercícios

Personalização de Veículos Futuristas

Uma empresa desenvolve **veículos futuristas** que podem ser configurados para diferentes **ambientes**:

- **Terra** → Veículos como **Carros Voadores e Motos Autônomas**
- **Espaço** → Veículos como **Naves e Exploradores Robóticos**

Cada veículo tem dois aspectos principais:

1. **Modo de Propulsão** (ex: Motor a jato, Plasma, etc.)
2. **Sistema de Controle** (ex: Inteligência Artificial, Controle Manual, etc.)

Continua ->

Exercícios

Tarefa

1. Modele uma **Abstract Factory** que permita criar **veículos futuristas completos** (propulsão + sistema de controle).
2. Crie duas fábricas concretas: **Fábrica de Veículos para Terra** e **Fábrica de Veículos para Espaço**.
3. No código cliente, crie uma função que **permita configurar um veículo futurista escolhendo a fábrica correta**.

Código fonte

https://github.com/paeeglee/patterns