# Sistemas de Informação

## Arquitetura e Padrões de Projeto

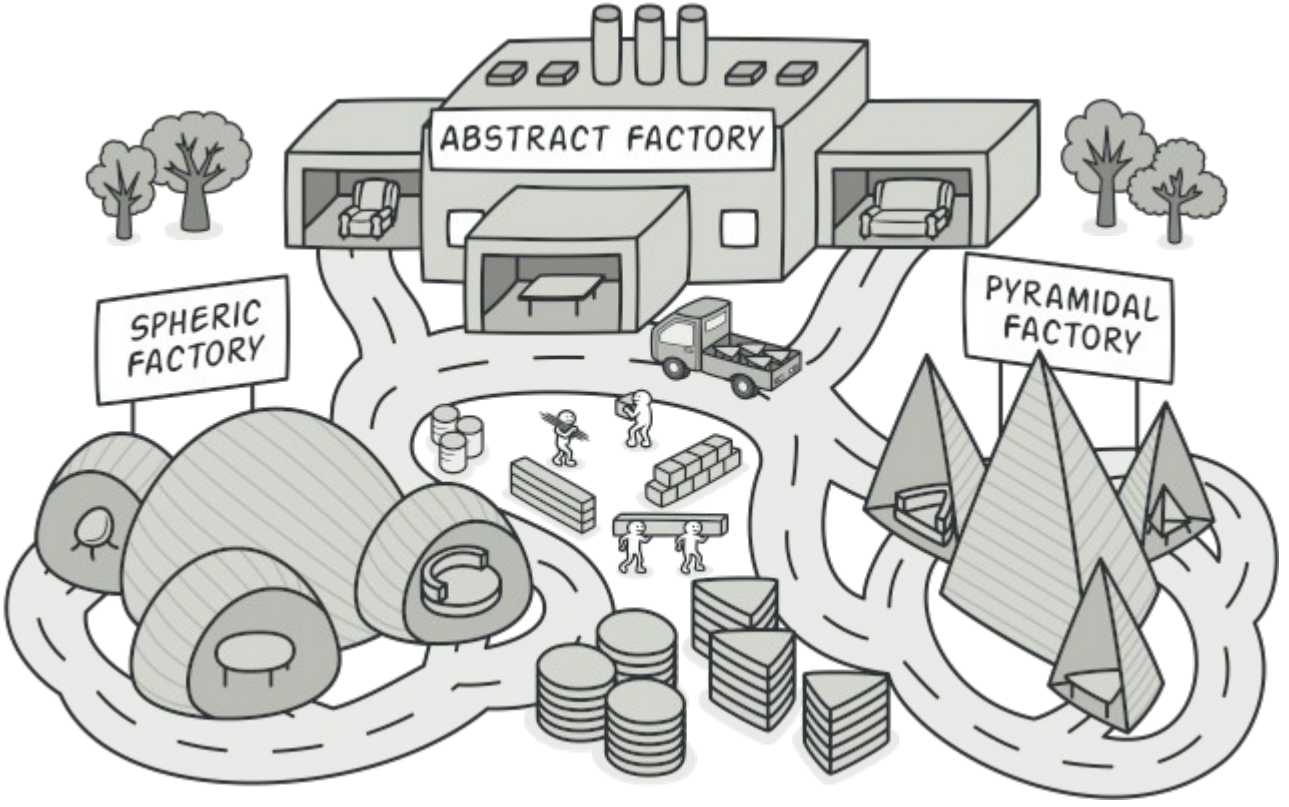
Prof. Erik Aceiro Antonio

06 – 04 – 2020
19h30m

# Padrões

- Princípios de Projetos Orientados a Objetos
  - SOLID (
- Padrões anteriores
  - Singleton
  - Façade
- Padrões dessa aula
  - Abstract Factory
  - Factory Method

# Abstract Factory



# Problema

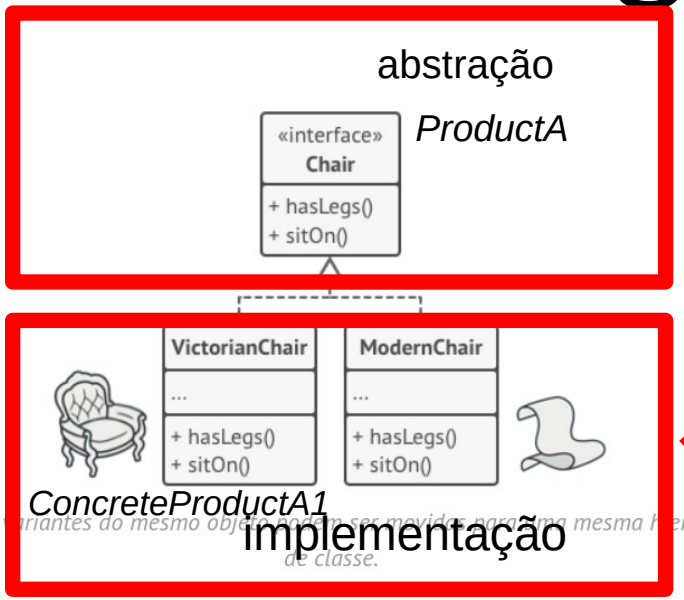
Imagine que você está criando um simulador de loja de mobílias. Seu código consiste de classes que representam:

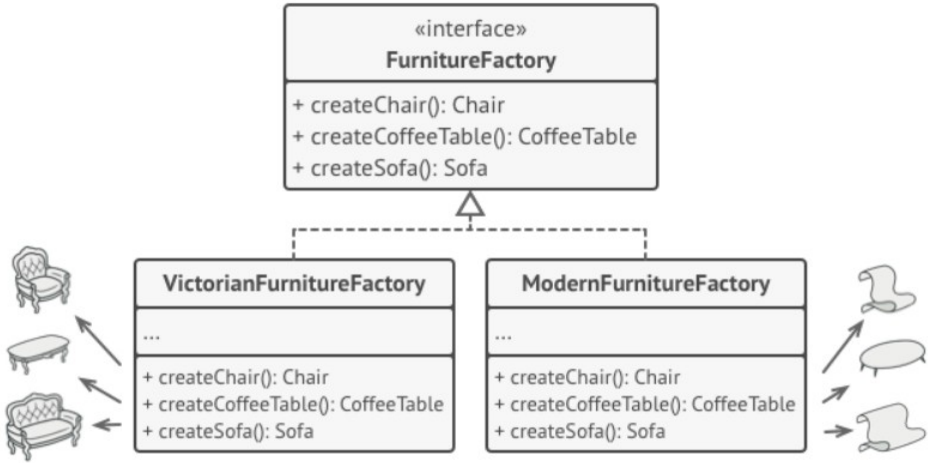
1. Uma família de produtos relacionados, como: `Cadeira` + `Sofá` + `MesaDeCentro`.

2. Várias variantes dessa família. Por exemplo, produtos `Cadeira` + `Sofá` + `MesaDeCentro` estão disponíveis nessas variantes: `Moderno`, `Vitoriano`, `ArtDeco`.


*Um sofá no estilo Moderno não combina com cadeiras de estilo Vitoriano*

# Solução




*Cada fábrica concreta corresponde a uma variante de produto específica.*

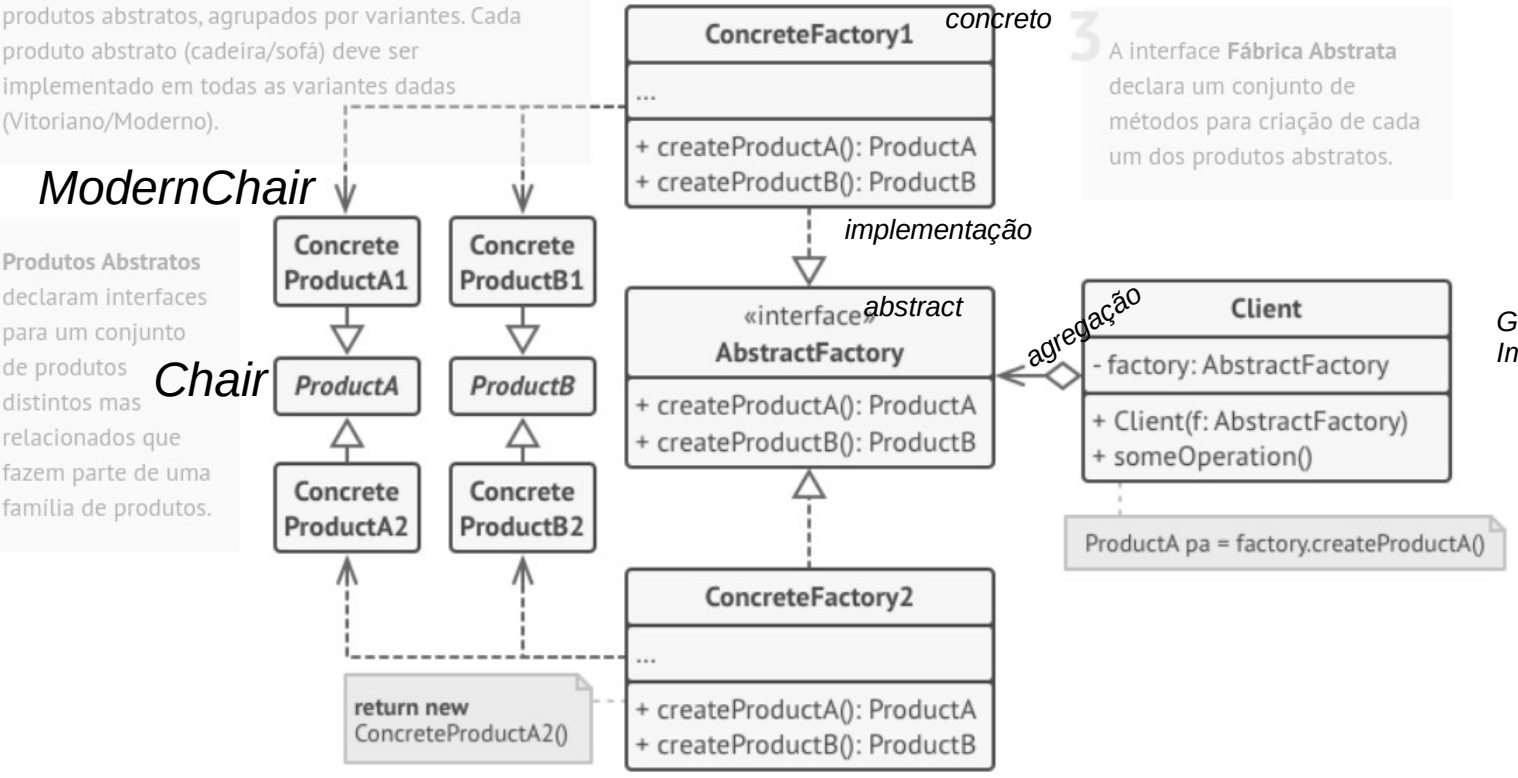
O próximo passo é declarar a *fábrica abstrata*—uma interface com uma lista de métodos de criação para todos os produtos que fazem parte da família de produtos (por exemplo, `criarCadeira`, `criarSofá` e `criarMesaDeCentro`). Esses métodos devem retornar tipos **abstratos** de produtos representados pelas interfaces que extraímos previamente: `Cadeira`, `Sofá`, `MesaDeCentro` e assim por diante.

qual a técnica de programação usada para desacoplar a abstração da implementação ?

→ **overriding** – substituindo ou sobrescrevendo o conteúdo do método
→ **LSP – Liskov Substitution Principle – Polimorfismo**
→ **ISP – Interface Segregate Principle – separa a abstração da implementação**

2 **Produtos Concretos** são várias implementações de produtos abstratos, agrupados por variantes. Cada produto abstrato (cadeira/sofá) deve ser implementado em todas as variantes dadas (Vitoriano/Moderno).

1 **Produtos Abstratos** declaram interfaces para um conjunto de produtos distintos mas relacionados que fazem parte de uma família de produtos.




3 A interface **Fábrica Abstrata** declara um conjunto de métodos para criação de cada um dos produtos abstratos.

4 **Fábricas Concretas** implementam métodos de criação fábrica abstratos. Cada fábrica concreta corresponde a uma variante específica de produtos e cria apenas aquelas variantes de produto.

5 Embora fábricas concretas instanciam produtos concretos, assinaturas dos seus métodos de criação devem retornar produtos *abstratos* correspondentes. Dessa forma o código cliente que usa uma fábrica não fica ligada a variante específica do produto que ele pegou de uma fábrica. O **Cliente** pode trabalhar com qualquer variante de produto/fábrica concreto, desde que ele se comunique com seus objetos via interfaces abstratas.

# Aplicação

**Use o Abstract Factory quando seu código precisa trabalhar com diversas famílias de produtos relacionados, mas que você não quer depender de classes concretas daqueles produtos-eles podem ser desconhecidos de antemão ou você simplesmente quer permitir uma futura escalabilidade.**

O Abstract Factory fornece a você uma interface para a criação de objetos de cada classe das famílias de produtos. Desde que seu código crie objetos a partir dessa interface, você não precisará se preocupar em criar uma variante errada de um produto que não coincida com produtos já criados por sua aplicação.

- Considere implementar o Abstract Factory quando você tem uma classe com um conjunto de **métodos fábrica** que desfoquem sua responsabilidade principal.

- Em um programa bem desenvolvido *cada classe é responsável por apenas uma coisa*. Quando uma classe lida com múltiplos tipos de produto, pode valer a pena extrair seus métodos fábrica em uma classe fábrica solitária ou uma implementação plena do Abstract Factory.

## ⚖ Prós e contras

- ✓ Você pode ter certeza que os produtos que você obtém de uma fábrica são compatíveis entre si.
- ✓ Você evita um vínculo forte entre produtos concretos e o código cliente.
- ✓ *Princípio de responsabilidade única*. Você pode extrair o código de criação do produto para um lugar, fazendo o código ser de fácil manutenção.
- ✓ *Princípio aberto/fechado*. Você pode introduzir novas variantes de produtos sem quebrar o código cliente existente.
- ✗ O código pode tornar-se mais complicado do que deveria ser, uma vez que muitas novas interfaces e classes são introduzidas junto com o padrão.

# ⇄ Relações com outros padrões

- Muitos projetos começam usando o **Factory Method** (menos complicado e mais customizável através de subclasses) e evoluem para o **Abstract Factory**, **Prototype**, ou **Builder** (mais flexíveis, mas mais complicados).

- O **Builder** foca em construir objetos complexos passo a passo. O **Abstract Factory** se especializa em criar famílias de objetos relacionados. O *Abstract Factory* retorna o produto imediatamente, enquanto que o *Builder* permite que você execute algumas etapas de construção antes de buscar o produto.

- Classes **Abstract Factory** são quase sempre baseadas em um conjunto de **métodos fábrica**, mas você também pode usar o **Prototype** para compor métodos dessas classes.

- O **Abstract Factory** pode servir como uma alternativa para o **Facade** quando você precisa apenas esconder do código cliente a forma com que são criados os objetos do subsistema.

- Você pode usar o **Abstract Factory** junto com o **Bridge**. Esse pareamento é útil quando algumas abstrações definidas pelo *Bridge* só podem trabalhar com implementações específicas. Neste caso, o *Abstract Factory* pode encapsular essas relações e esconder a complexidade do código cliente.

- As **Fábricas Abstratas**, **Construtores**, e **Protótipos** podem todos ser implementados como **Singletons**.

# Exercício

- Acesse o endereço
  - https://refactoring.guru/pt-br/design-patterns/abstract-factory
- Construia o exemplo do ChairFactory visto em aula em Python