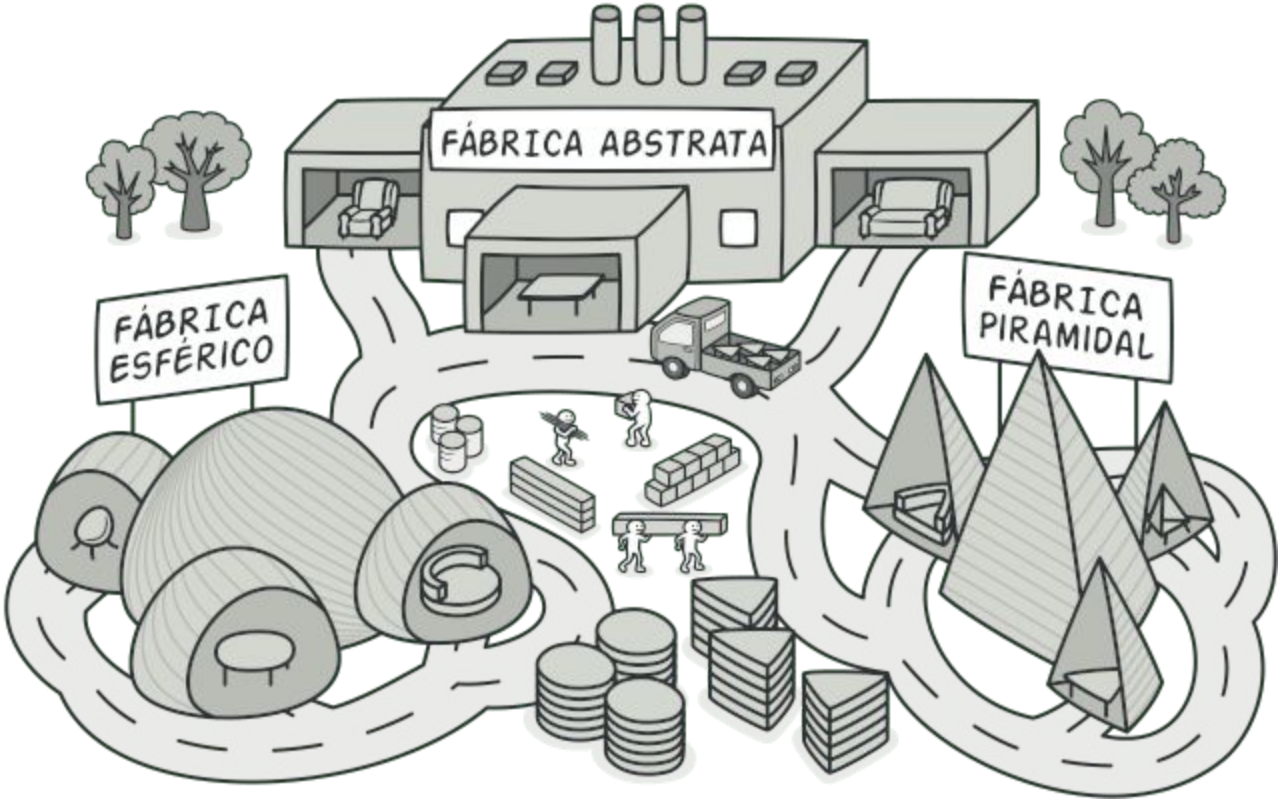
Abstract Factory

Equipe: Luís Clício, Armando, Hudson, Adriano


FÁBRICA ABSTRATA
FÁBRICA ESFÉRICO
FÁBRICA PIRAMIDAL

Propósito 

O Abstract Factory é um padrão de projeto criacional que permite que você **produza famílias de objetos relacionados** sem ter que especificar suas classes concretas.

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*.

Problema

	Cadeira	Sofá	Mesa de Centro
Art Deco			
Vitoriano			
Moderno			

Problema

Você precisa de um jeito de criar objetos de mobília individuais para que eles combinem com outros objetos da mesma família. Os clientes ficam muito bravos quando recebem mobília que não combina.

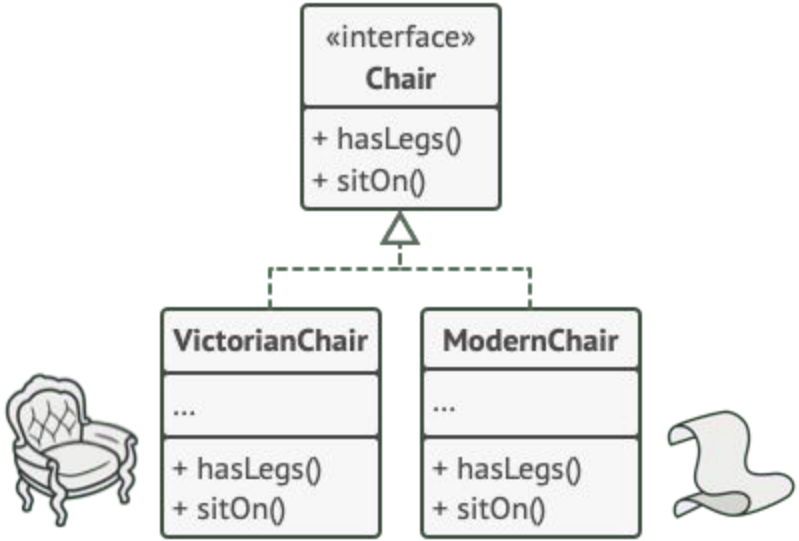
Problema 

E ainda, você **não quer mudar o código existente** quando adiciona novos produtos, ou famílias de produtos ao programa. Os vendedores de mobílias atualizam seus catálogos com frequência e você não vai querer mudar o código base cada vez que isso acontece.

Solução 

A primeira coisa que o padrão Abstract Factory sugere é declarar explicitamente **interfaces para cada produto distinto da família** de produtos (ex: cadeira, sofá ou mesa de centro). Então você pode fazer todas as **variantes dos produtos seguirem essas interfaces**. Por exemplo, todas as variantes de cadeira podem implementar a interface `*Cadeira*`; todas as variantes de mesa de centro podem implementar a interface `*MesaDeCentro*`, e assim por diante.

Solução


«interface»
Chair
+ hasLegs()
+ sitOn()
VictorianChair
...
+ hasLegs()
+ sitOn()
ModernChair
...
+ hasLegs()
+ sitOn()

Solução 

Agora, e o que fazer sobre as variantes de produtos? **Para cada variante** de uma família de produtos nós **criamos uma classe fábrica** separada baseada na interface *FábricaAbstrata*. Uma fábrica é uma classe que retorna **produtos de um tipo** em particular. Por exemplo, a classe *FábricaMobíliaModerna* só pode criar objetos *CadeiraModerna*, *SofáModerno*, e *MesaDeCentroModerna*.

Exemplo da estrutura

Interfaces dos produtos

```python
class AbstractProductA(abc.ABC):
    @abc.abstractmethod
    def method_a(self) -> str:
        pass
```

```python
class AbstractProductB(abc.ABC):
    @abc.abstractmethod
    def method_b(self) -> str:
        pass

    @abc.abstractmethod
    def another_method_b(self) -> str:
        pass
```

Interface da fábrica abstrata

```python
class AbstractFactory(abc.ABC):
    @abc.abstractmethod
    def create_product_a(self) -> AbstractProductA:
        pass

    @abc.abstractmethod
    def create_product_b(self) -> AbstractProductB:
        pass
```

Produtos ‘A’ concretos

```python
class ConcreteProductA1(AbstractProductA):
    def method_a(self):
        return 'Product A1'
```

```python
class ConcreteProductA2(AbstractProductA):
    def method_a(self):
        return 'Product A2'
```

Produto ‘B1’ concreto

```python
class ConcreteProductB1(AbstractProductB):
    def method_b(self):
        return 'Product B1'

    def another_method_b(self):
        return 'Product B1 another method'
```

Produto ‘B2’ concreto

```python
class ConcreteProductB2(AbstractProductB):
    def method_b(self):
        return 'Product B2'

    def another_method_b(self):
        return 'Product B2 another method'
```

Fábrica concreta 1

```python
class ConcreteFactory1(AbstractFactory):
    def create_product_a(self):
        return ConcreteProductA1()

    def create_product_b(self):
        return ConcreteProductB1()
```

Fábrica concreta 2

```python
class ConcreteFactory2(AbstractFactory):
    def create_product_a(self):
        return ConcreteProductA2()

    def create_product_b(self):
        return ConcreteProductB2()
```

Código cliente

```python
def client_code(factory: AbstractFactory):
    product_a = factory.create_product_a()
    product_b = factory.create_product_b()

    print(product_a.method_a())
    print(product_b.method_b())
```