| | |
|---|---|
| **Curso:** Sistemas de Informação | |
| **Disciplina:** Padrões de projeto | **Período:** 8 |
| **Professor:** Luiz Gustavo Dias | **Tipo:** Design Patterns |
| **Objetivo:** Abstract Factory | |

## DESIGN PATTERNS: ABSTRACT FACTORY

### Intenção

Este padrão permite a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe concreta seja especificada. Uma factory é a localização de uma classe concreta no código em que objetos são construídos.

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.

O uso deste padrão torna possível trocar implementações concretas sem alterar o código que estas usam, mesmo em tempo de execução. No entanto, o emprego deste padrão, como acontece com outros padrões semelhantes, pode resultar em uma complexidade desnecessária e trabalho extra no início do código. Além disso, os níveis mais elevados de abstração podem resultar em sistemas que são mais difíceis de manter.

A essência do padrão Abstract Factory é fornecer uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

### Sobre o Padrão

- É um padrão de criação, portanto lida com a criação de objetos;
- É uma fábrica, assim como o Factory Method e geralmente é composto por múltiplos Factory Methods;
- Visa agrupar famílias de produtos compatíveis, criando uma fábrica concreta por grupo de objetos compatíveis;

- Separa o código que cria do código que usa (Single Responsability Principle);
- Permite a fácil implementação de novas famílias de objeto (Open/Closed Principle);
- Toda a programação fica focada nas interfaces e não em implementações

**Aplicabilidade**

- um sistema deve ser independente de como seus produtos são criados, compostos ou representados;
- um sistema deve ser configurado como um produto de uma família de múltiplos produtos;
- uma família de objetos-produto for projetada para ser usada em conjunto, e você necessita garantir esta restrição;
- você quer fornecer uma biblioteca de classes de produtos e quer revelar somente suas interfaces, não suas implementaçõ

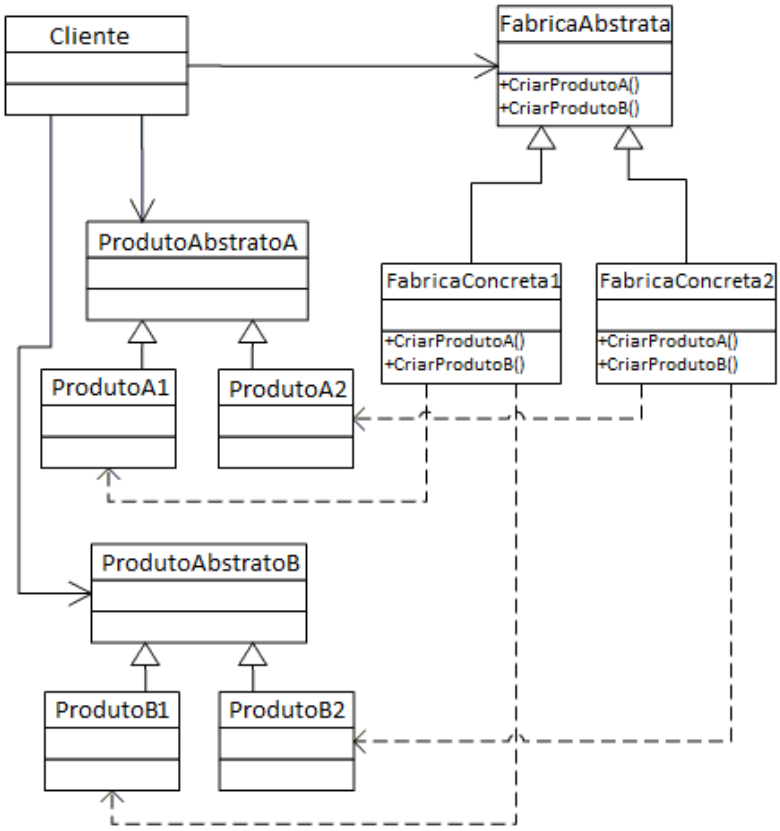
O padrão Abstract Factory pode ser utilizado na implementação de um toolkit que disponibilize controles que funcionem em diferentes interfaces gráficas, tal como Motif, GTK+ (GNOME) ou Qt (KDE). Estas GUIs possuem diferentes padrões de controles visuais e, para facilitar a construção de aplicativos que interajam facilmente com diferentes interfaces gráficas, é interessante que se defina interfaces comuns para acesso aos controles, independentemente da GUI utilizada.

Este problema pode ser resolvido por meio de uma classe abstrata que declara uma interface genérica para criação dos controles visuais e de uma classe abstrata para criação de cada tipo de controle. O comportamento específico, de cada um dos padrões tecnológicos contemplados, é implementado por meio de uma classe concreta. O aplicativo, ou "cliente", interage com o toolkit por meio das classes abstratas sem ter conhecimento da implementação das classes concretas.

Um exemplo bem simplista seria um projeto com interface para Mobile e para Desktop, uma boa opção para reaproveitar os mesmos controles de interface seria criar pacotes com classes abstratas e os pacotes com as classes concretas implementando apenas as diferenças.

Esse padrão também se aplica na padronização de ambientes, por exemplo, tamanhos de botões, fontes, cores de fundo, largura de bordas. Com isso e havendo uma política que exija que os desenvolvedores usem essas classes em vez das nativas da linguagem, ajudará a padronizar a aparência e comportamento das aplicações.

**Estrutura**




**Fábrica Abstrata**

Pode ser uma classe abstrata ou uma interface, mas a classe abstrata é utilizada com maior frequência. Seu objetivo é declarar métodos de criação de objetos do tipo ProdutoAbstrato, que são implementados por uma classe do tipo FabricaConcreta, que estende ou implementa a FabricaAbstrata.

**Produto Abstrato**

Pode ser uma classe abstrata ou uma interface, mas a classe abstrata é utilizada com maior frequência. Produto abstrato declara os métodos que são implementados por classes do tipo ProdutoConcreto. FabricaConcreta cria internamente um objeto do tipo ProdutoConcreto, mas esse objeto é retornado como um ProdutoAbstrato. O Abstract Factory não sabe qual ProdutoConcreto está sendo criado, mas sabe quais métodos do produto ele pode utilizar.

**Fábrica Concreta**

Estende ou implementa a FabricaAbstrata. O objetivo dessa classe é implementar os métodos declarados em FabricaAbstrata, criando um objeto do tipo ProdutoConcreto e retornando-o como um ProdutoAbstrato. Isso é polimorfismo. É comum existir mais de uma classe do tipo ProdutoConcreto assim como ocorre com FabricaConcreta. A quantidade de classes do tipo FabricaConcreta está diretamente ligada com a quantidade de classes do tipo ProdutoConcreto.

**Produto Concreto(ProdutoA1, ProdutoA2, etc..)**

Estende ou implementa a classe ProdutoAbstrato. Nessa classe são implementados os métodos declarados em ProdutoAbstrato. Essa é a classe que faz uma instância concreta ser criada. Para cada FabricaConcreta, há pelo menos um ProdutoConcreto.

**Consequências**

**Positivas:**

1. Os produtos sempre serão compatíveis entre si;

2. Aplicação clara do Open/Closed Principle – é fácil adicionar novas fábricas de produtos;

3. Aplicação clara do Single Responsability Principle – o código que cria está separado do código que usa o objeto.

**Negativas:**

1. Muitas classes e maior complexidade no código.