PADRÕES DE PROJETO

- **ABSTRACT FACTORY**

❖ Definição:

O padrão de projeto Abstract Factory, ou Fábrica Abstrata, é um dos padrões Criacionais com o foco na criação de objetos que ajudam na flexibilização e reutilização de código.

Dentre eles temos os:

- Factory Method;
- Abstract Factory;
- Builder;
- Protorype;
- Singleton

Referente ao método Abstract ele tem como função a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem invocar a classe concreta. Em outras palavras, é como se existisse uma classe-pai contendo características em comum dentre suas subcategorias (filhos).

Fábrica de fábricas que se dividem em categorias.

- Histórico:

É importante ressaltar que a origem dos Design Patterns (Padrões de Projeto) provem do trabalho de um arquiteto chamado Christopher Alexander, no final da década de 70 e esses Padrões de Projeto estabelecem técnicas genéricas para solucionar problemas recorrentes de um determinado desenvolvimento.

Com o tempo, muitos padrões foram criados dentre eles 23 foram formulados por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides em 1995.

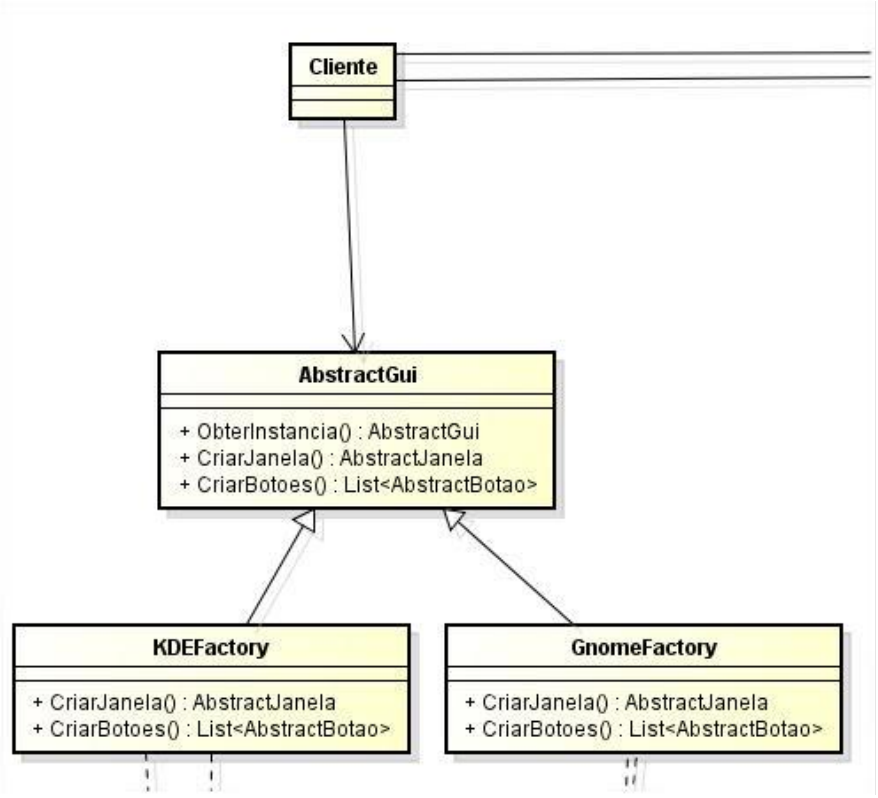
- Utilização:

Normalmente usados quando seu código precisa trabalhar com diversas famílias de produtos relacionados, mas que você não quer que eles dependam de classes concretas que é uma classe que possui atributos, métodos construtores e outros e pode ser instanciada, ou seja, permite a criação de objetos a partir dela.

Assim com o Abstract Factory você tem uma abstração dessas classes transformando tudo em uma interface onde lá vai conter os métodos em comum.

Para casos em que os objetos têm pequenas diferenças entre eles vamos ter classes especificas daquela família que vai herdar a nossa interface e a implementar com modificações.

- Exemplos de Utilização:



Nesse caso a classe cliente solicita uma janela, mas nessa aplicação ele não precisa saber se é uma janela da KDE ou da Gnome e para isso fazemos um abstract factory que vai ter inicialmente os métodos de criação de janela e etc.

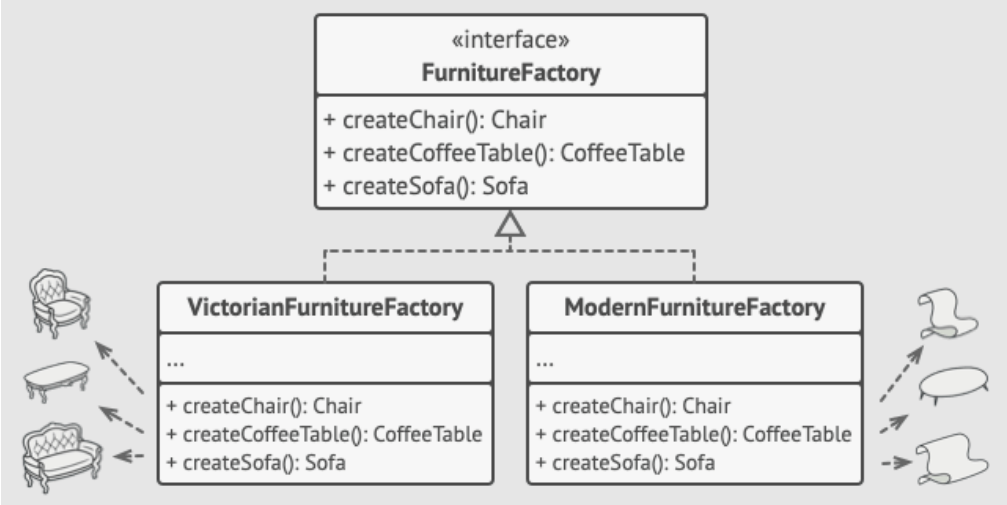
E cada uma das marcas abaixo vai herdar isso e poderá sobrescrever os métodos para adequar a janela base ao seu tipo de layout de janela.

- Outro Exemplo:

Uma empresa vende mesas cadeiras e sofás essa mesma empresa produz esses produtos em três versões diferentes.

O cliente quando ele comprar ele não vai querer um produto de um estilo diferente do outro porque eles não combinam.

A primeira coisa a ser feita é no padrão Abstracty Factory vai ser a criação de uma interface abstrata onde podemos ter os métodos de construir as cadeiras, as mesas e etc.



Essa interface vai ser aplicada individualmente para cada Fábrica responsável por um estilo, que vai poder sobrescrever os métodos para construir os móveis por exemplo em um estilo moderno.

- ❖ Mercado de Trabalho:

Analisando o atual cenário da tecnologia e do desenvolvimento de softwares, visando maior eficiência e utilizando metodologias ágeis, um padrão de projeto como o abstract factory é necessária, pois utilizam soluções e arquiteturas bem-sucedidas elaboradas no passado para evitar e resolver problemas recorrentes ou repetitivos de um contexto, permitindo o reaproveitamento de códigos, estruturas e soluções e permitindo também a otimização de tempo e economia de recursos.

Por fim, o padrão Abstract Factory é muito usado no mercado na implementação de toolkits – que no geral são um conjunto de interfaces gráficas de uma GUI (Interface Gráfica do Utilizador).

- ❖ Padrões Similares:

O Abstract Factory é um padrão de criação e pode se relacionar com outros padrões semelhantes, como:

- Builder : O *Abstract Factory* retorna o produto imediatamente, enquanto o *Builder* permite que você execute algumas etapas de construção passo a passo antes de buscar o produto;
- Prototype: Pode compor métodos das classes Abstract Factory;
- Facade: 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 sistema;
- Bridge: o *Abstract Factory* pode encapsular abstrações definidas pelo *Bridge* e esconder a complexidade do código cliente;

- **ADAPTER**

❖ Definição:

O Adapter é um padrão de projeto estrutural, no qual, possibilita objetos com interfaces incompatíveis colaborarem entre si.

O padrão Adapter atua como uma “ponte” entre duas interfaces. Este se envolve uma única classe chamada adaptador que é responsável pela comunicação entre duas interfaces independentes ou como dito, incompatíveis.

❖ Histórico:

Em 1978 os arquitetos Christopher Alexander, Sara Ishikawa e Murray Silverstein escreveram um livro chamado “A Pattern Language: Towns, Buildings, Construction”.Neste livro os autores catalogaram 253 tipos de problemas (ou desafios de projeto) e analisaram o que está por trás de cada situação, descrevendo-as na sua essência e propondo uma solução padrão.

Mas esses problemas e conceitos ficaram realmente conhecidos em 1994, quando os engenheiros de software Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides escreveram o livro “Design Patterns: Elements of Reusable Object-Oriented Software” com o objetivo de catalogar problemas comuns aos projetos de desenvolvimento de software e as formas de resolver esses problemas. Os autores catalogaram 23 padrões que utilizaram ao longo de suas carreiras.

Desde então, Design Patterns tem sido um tema bastante estudado por programadores e arquitetos de software pelo mundo todo.

❖ Utilização:

- É utilizado quando temos uma classe existente cuja interface não é adequada para as suas necessidades;
- Permite que classes que não possuem uma interface comum trabalhem de forma conjunta;

❖ Exemplos de Utilização:

```
public class TomadaDeDoisPinos {
    public void ligarNaTomadaDeDoisPinos() {
        System.out.println("Ligado na Tomada de Dois Pinos");
    }
}

public class TomadaDeTresPinos {
    public void ligarNaTomadaDeTresPinos() {
        System.out.println("Ligado na Tomada de Tres Pinos");
    }
}

public class AdapterTomada extends TomadaDeDoisPinos {
    private TomadaDeTresPinos tomadaDeTresPinos;

    public AdapterTomada(TomadaDeTresPinos tomadaDeTresPinos) {
        this.tomadaDeTresPinos = tomadaDeTresPinos;
    }

    public void ligarNaTomadaDeDoisPinos() {
        tomadaDeTresPinos.ligarNaTomadaDeTresPinos();
    }
}
```

Veja que nós temos uma classe TomadaDeDoisPinos mas nós queremos nos conectar a uma classe TomadaDeTresPinos que possui outros métodos e uma outra interface diferente. Assim criamos um Adapter para que possamos acessá-la. Veja que o Adaptador herda da classe que você possui (o seu Target como mostrado no diagrama de classes). Dentro do adaptador temos o que o cliente precisa que é o TomadaDeTresPinos que será chamado posteriormente no método ligarNaTomadaDeDoisPinos que na verdade está chamando o método ligarNaTomadaDeTresPinos do novo fornecedor.

Para executarmos o aplicativo de teste podemos usar a implementação abaixo:

```
public class Teste {

    public static void main(String args[]) {
        TomadaDeTresPinos t3 = new TomadaDeTresPinos();

        AdapterTomada a = new AdapterTomada(t3);
        a.ligarNaTomadaDeDoisPinos();
    }

}
```

Nota-se que o cliente faz uma chamada normalmente usando a tomada de dois pinos, mas na realidade está chamada está sendo adaptada para uma tomada de três pinos. Assim temos duas interfaces que não eram compatíveis entre si conversando normalmente.

- Padrões Similares:

O padrão GoF Bridge possui uma estrutura similar, mas esse padrão tem uma intenção diferente: tem como objetivo separar uma interface da sua implementação, de uma forma que elas possam variar fácil e independentemente

Como por exemplo: O adapter vai receber um monte de funções e por trás vai executar outras fazendo o papel de adaptador mesmo. Já o Bridge é um padrão de design estrutural que permite dividir uma classe grande ou um conjunto de classes estreitamente relacionadas em duas hierarquias separadas — abstração e implementação — que podem ser desenvolvidas independentemente uma da outra

- Mercado de trabalho:

Com o objetivo de otimizar a produção de código, o mercado de trabalho atual apresenta aplicações de Adapters em muitos aspectos de desenvolvimento de software, especialmente em áreas que apresentam interação entre mais de um serviço/software ou manufaturador, o que pode gerar incompatibilidade se programados sem a adaptação de interfaces de comportamento similar, mas métodos diferentes. No geral, é muito utilizado por empresas de engenharia de software que trabalham com modelos diferentes de equipamentos, ou respostas de requisições em formatos diferentes.

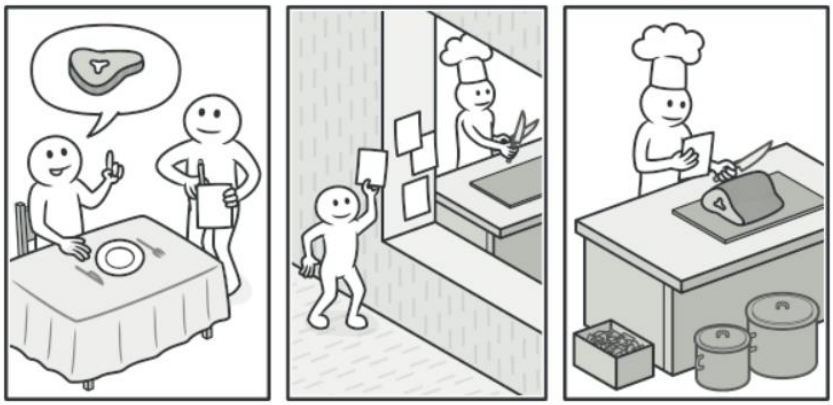
- **COMMAND**

❖ Definição:

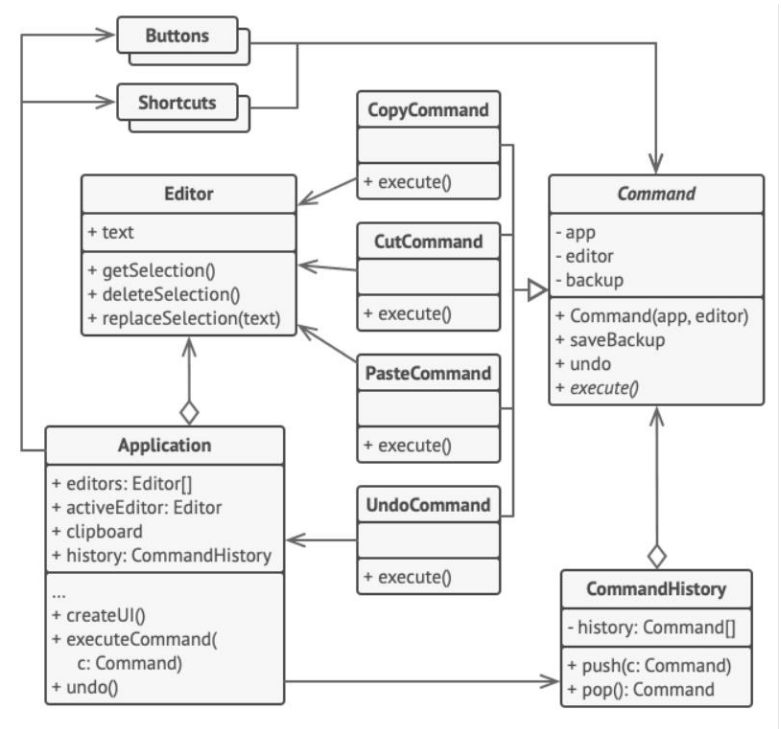
É um dos 11 padrões comportamentais dentre os 23 padrões de projeto, que transforma um pedido em um objeto independente que contém toda a informação sobre o pedido.

Essa transformação permite que você parametrize métodos com diferentes pedidos, filas ou requisições de Log, com suporte a reverter as operações.

❖ Analogia:



❖ Exemplo:



- Histórico – Breve contexto histórico

A origem dos Padrões de Projetos (Command que está no Padrões de Projeto) surgiu no trabalho de um arquiteto chamado Christopher Alexander que escreveu dois livros de bastante sucesso onde ele exemplificava o uso e descrevia seu raciocínio para documentar os padrões para a arquitetura (como portas, janelas, etc).

- Para que serve – Utilização, onde se utiliza?

Se usa o Command em várias situações como:

-Usados para implementação

-Parametrizar objetos por uma ação a ser executada

-Especificar, enfileirar e executar solicitações em tempos diferentes

-Dar suporte para desfazer operações

-Estruturar um sistema em torno de operações de alto nível (Transações)

-Reduzir acoplamento entre as requisições dos clientes e os objetos que as executam

-Facilitar a implantação de novas operações e tornar mais simples a manutenção das operações

- Mercado de Trabalho – Empresas que utilizam:

Como mostrado antes, o Command faz parte dos Padrões de Projeto, e no mercado de trabalho você irá encontrar empresas que usam esse tipo de padrão.

Como a:

-Triyo

-Infinitedev