**Padrão de Projeto: Strategy**

**O que é?**

O padrão de projeto Strategy é um padrão de design de software que se enquadra na categoria de padrões comportamentais. Ele permite definir uma família de algoritmos, encapsular cada um deles e torná-los intercambiáveis. Isso permite que o algoritmo varie independentemente dos clientes que o utilizam.

Em termos simples, o padrão Strategy é útil quando você tem várias maneiras de executar uma tarefa e quer ser capaz de escolher qual abordagem utilizar dinamicamente, durante a execução do programa, sem que isso afete o código que o utiliza.

No padrão Strategy, geralmente há três principais componentes:

**Contexto**: É a classe que possui uma referência para a estratégia escolhida. É responsável por interagir com o cliente e chamar a estratégia apropriada para executar a tarefa.

**Interface Strategy**: Define um contrato para os algoritmos suportados. Todos os algoritmos devem seguir essa interface, garantindo que cada um tenha métodos comuns para serem chamados pelo contexto.

**Estratégias Concretas**: São as implementações específicas dos algoritmos definidos na interface Strategy.

O benefício principal do padrão Strategy é a flexibilidade que oferece, permitindo que novas estratégias sejam adicionadas facilmente sem modificar muito o código existente. Isso promove um código mais modular, de fácil manutenção e extensão.

**Como funciona?**

O padrão de projeto Strategy é implementado com base em uma estrutura que permite a troca dinâmica de algoritmos durante a execução de um programa. Aqui está uma explicação passo a passo de como ele funciona:

**Identificação do Problema**: O padrão Strategy é utilizado quando há diferentes algoritmos para realizar uma tarefa específica e é desejável poder alternar entre esses algoritmos durante a execução do programa.

**Criação da Interface Strategy**: Começa-se definindo uma interface (ou classe abstrata) que declara os métodos que todos os algoritmos suportados devem implementar. Isso estabelece um contrato para garantir que todos os algoritmos tenham métodos comuns.

**Implementação das Estratégias Concretas**: Para cada variação do algoritmo, são criadas classes concretas que implementam a interface Strategy. Cada classe representa uma estratégia específica.

**Contexto**: É criada uma classe chamada "Contexto", que possui um campo para armazenar uma referência à interface Strategy. Essa classe é responsável por chamar os métodos da estratégia selecionada.

**Troca Dinâmica da Estratégia**: Durante a execução do programa, a estratégia a ser utilizada é definida dinamicamente, seja por meio de parâmetros, configurações ou lógica interna. O Contexto pode alterar a estratégia que está usando em tempo de execução.

**Utilização**: O cliente interage com o Contexto, que por sua vez invoca os métodos da estratégia selecionada para realizar a tarefa desejada.

**Flexibilidade e Extensibilidade**: Como diferentes estratégias podem ser facilmente adicionadas sem alterar o código do Contexto ou do cliente, o padrão Strategy oferece flexibilidade e facilidade de manutenção.

Em resumo, o padrão Strategy separa os algoritmos das classes que os utilizam, permitindo a seleção e troca de estratégias de maneira dinâmica, sem modificar o código existente e mantendo a coesão e flexibilidade do sistema.

**Qual problema resolve?**

O padrão Strategy resolve o problema de ter múltiplas maneiras de realizar uma tarefa e permite trocar entre essas abordagens dinamicamente durante a execução do programa, sem modificar o código cliente. Ele promove flexibilidade ao separar algoritmos em classes separadas e intercambiáveis.