Strategy

Arthur Maia, Bruno Moreira Lima, Claudio Pales Costa, Gabriel Honorato Santos Ferraz, Kéven Patricio

[1]Centro Universitário de Excelência de Vitória da Conquista (UNEX)
Sistema de informação

arturcoqueiro018@gmail.com, bru.no@outlook.com.br,

claudio.palles.costa@gmail.com, ferrazcoelhorodrigo@gmail.com,

patriciokeven4@gmail.com

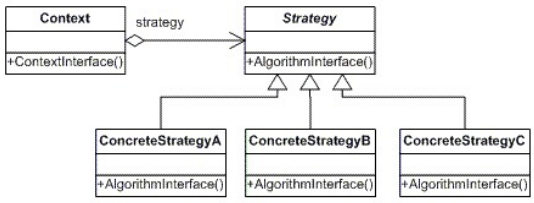
1. Introdução

A refatoração visa implementar funções novas em um código já existente, mas sem ter que modificar o código por inteiro. Ela melhora o design do sistema sem impactar o usuário final. Nesse contexto, também entram os Padrões de Projeto, que transformam um design mediano em algo bem mais coeso e adaptável. Para isso, implementaremos o Padrão Strategy, que irá colocar cada algoritmo em sua própria classe, permitindo que o sistema de relatórios troque o algoritmo de forma flexível.

2. Fundamentação Teórica

O Padrão Strategy resolve o problema de acomplamento excessivo que é calsado quando uma classe precissa gerecenciar múltiplas variações de um mesmo algoritmo, assim padrão strategy surge para eliminar essa rigidez, permitindo que os algoritmos de relatório sejam substituidos e que a classe principal apenas permita que a execução para a strategy selecionada, transformando o codigo em algo bem mais limpo e compliensivel.

3. diagrama de classes



4. Desvantagens do Padrão Strategy

A principal desvantagem do padrão Strategy é que ele aumenta o overhead das classes, pois exige que cada variação de algoritmo seja encapsulada em uma classe concreta separada. Também aumenta a complexidade das configurações. Em vez de simplesmente chamar um método na classe principal, as classes agora precisam ser configuradas com o objeto Strategy correto.

5. Metodologia

A refatoração do módulo de relatórios utilizou o Padrão Strategy para garantir maior manutenibilidade e extensibilidade ao sistema. O processo de desenvolvimento seguiu uma sequência lógica para isolar os algoritmos de formatação.

Passo a Passo do Desenvolvimento Identificação do Problema A análise do sistema demonstrou que o método anterior de geração de relatórios dependia de estruturas condicionais (if/else ou switch) em uma única classe para selecionar o formato de saída. Esse design era rígido e violava o Princípio Open/Closed (OCP), pois qualquer novo tipo de relatório exigiria a modificação do código central.

Modelagem do Padrão Strategy O Padrão Strategy foi selecionado para resolver o problema, pois permite que o algoritmo seja trocado dinamicamente. A modelagem envolveu a definição dos três componentes essenciais:

Strategy (Interface): Criação da interface RelatorioStrategy para definir o contrato gerarRelatorio().

Concrete Strategies: Implementação das classes RelatorioSimplificadoStrategy e RelatorioDetalhadoStrategy. Cada classe encapsulou toda a lógica de cálculo e formatação específica do seu respectivo relatório.

Refatoração do Código Cliente O código que anteriormente continha a lógica de seleção de relatórios foi refatorado para se tornar o Contexto. O Contexto foi modificado para receber uma instância da RelatorioStrategy como dependência, delegando a chamada do método gerarRelatorio() à estratégia selecionada.

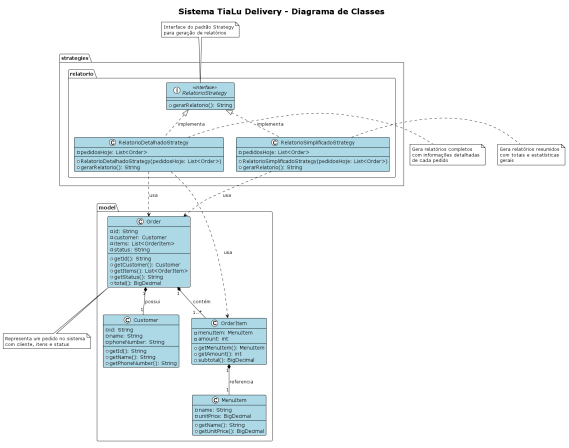
Essa abordagem garantiu o isolamento total entre a lógica do cliente e as implementações de relatórios, cumprindo o objetivo de design do projeto.

6. Resultados

A aplicação do Padrão Strategy resultou em um módulo de relatórios flexível e coeso.

Diagrama de Classes da Solução

O diagrama UML abaixo mostra a estrutura do Padrão Strategy aplicada à geração de relatórios:



A interface RelatorioStrategy define o método gerarRelatorio(), implementado por RelatorioSimplificadoStrategy e RelatorioDetalhadoStrategy.

Exemplo de Código // Interface (Strategy) public interface RelatorioStrategy String gerarRelatorio();

// Uso no cliente List¡Order¿ pedidos = Arrays.asList(/* dados... */);

RelatorioStrategy estrategia = new RelatorioSimplificadoStrategy(pedidos); System.out.println(estrategia.gerarRelatorio());

estrategia = new RelatorioDetalhadoStrategy(pedidos); System.out.println(estrategia.gerarRelatorio());

Vantagens

Extensível (OCP): Novos formatos podem ser adicionados sem alterar o código existente.

Manutenível: Cada relatório é isolado, facilitando ajustes.

Clareza: Elimina condicionais extensas (if/else, switch).

Desvantagens

Mais classes: Cada estratégia gera um novo arquivo.

Configuração explícita: O cliente precisa definir qual estratégia usar.

7. Considerações Finais

O projeto atingiu seu objetivo principal ao refatorar com sucesso o módulo de relatórios utilizando o Padrão Strategy. Essa solução foi fundamental para eliminar o acoplamento excessivo e a rigidez do sistema anterior, resultando em um código de relatórios significativamente mais extensível e manutenível. O design final adere diretamente ao Princípio Open/Closed (OCP), garantindo que novos formatos de relatório possam ser adicionados futuramente sem a necessidade de modificar o código já existente. Apesar da elegância e simplicidade inerentes ao padrão, a principal complexidade técnica do projeto se concentrou na lógica de negócios interna de cada estratégia de relatório. Isso exigiu um foco especial em dois pontos cruciais: o manuseio obrigatório de valores através do tipo BigDecimal, vital para assegurar a precisão correta dos cálculos financeiros, e a implementação de formatações específicas diretamente dentro de cada estratégia. Para continuar aprimorando o módulo, as próximas etapas sugeridas visam otimizar a funcionalidade e a arquitetura do sistema, incluindo a expansão do leque de opções de saída com a implementação de novas estratégias para formatos de dados como XML ou JSON, e a simplificação da configuração do código cliente e o gerenciamento das estratégias através da utilização de um framework para gerenciar a Injeção de Dependência, facilitando a configuração das estratégias no Contexto principal.