Escolha e descreva um padrão de projetos.

Padrão de Projeto Strategy

Intenção

Criar uma Strategy para cada variante e fazer com que o método delegue o algoritmo para uma instância de Strategy.

Outros nomes dado ao Padrão

Policy, Motivação

A lógica condicional é uma das estruturas mais complexas e utilizadas no desenvolvimento de softwares corporativos. Lógicas condicionais tendem a crescer e tornar-se cada vez mais sofisticadas, maiores e mais difíceis de manter com o passar do tempo. O padrão Strategy ajuda a gerenciar toda essa complexidade imposta pelas lógicas condicionais. O Padrão Strategy sugere que se produza uma família de classes para cada variação do algoritmo e que se forneça para a classe hospedeira uma instância de Strategy para a qual ela delegará em tempo de execução. Um dos pré-requisitos para o Strategy é uma estrutura de herança onde cada subclasse específica contém uma variação do algoritmo. Assim, o padrão Strategy possui diversos benefícios como clarificar algoritmos ao diminuir ou remover lógica condicional, simplificar uma classe ao mover variações de um algoritmo para uma hierarquia, e habilitar um algoritmo para ser substituído por outro em tempo de execução.

Aplicabilidade

Em resumo o padrão Strategy pode ser utilizado quando se tem as seguintes situações:

Quando muitas classes relacionadas diferem apenas no seu comportamento;

- Quando necessita-se de variantes de um algoritmo;
- Quando se precisa ocultar do usuário a exposição das estruturas de dados complexas, específicas do algoritmo;
- Quando uma classe define muitos comportamentos e por sua vez eles aparecem como diversos “IFs”. Com isso esses comandos condicionais são movidos para sua própria classe Strategy.

Estrutura

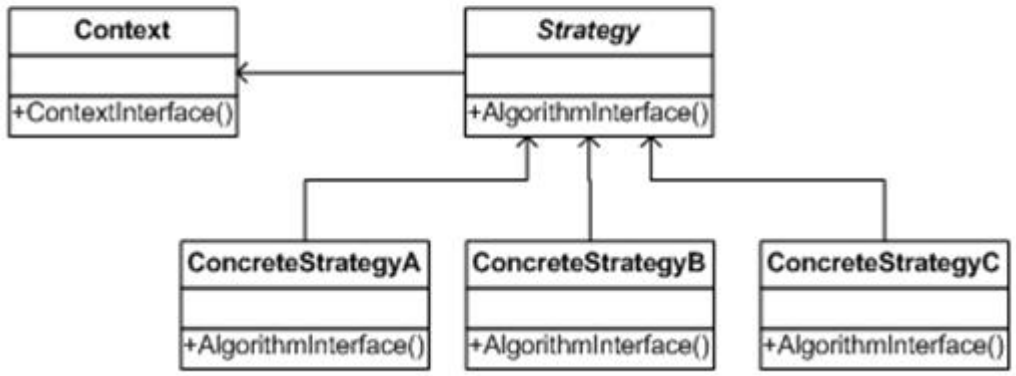

Figura 1: Exemplo da Estrutura do Padrão Strategy

Participantes

- **Strategy**: É uma interface comum para todas as subclasses, ou para todos os algoritmos que são suportados. O Contexto usa essa interface para chamar uma das subclasses ConcreteStrategy ou um dos algoritmos definidos.
- **ConcreteStrategy**: A classe concreta que herda da Strategy abstrata está definida como as subclasses ConcreteStrategyA, ConcreteStrategyB e ConcreteStrategyA no diagrama da **figura 1**.
- **Context**: É aquele que vai acessar um dos algoritmos das subclasses de interface Strategy.

Consequências

Segue os seguintes benefícios e desvantagens do padrão Strategy:

- Entre os benefícios do padrão Strategy pode-se citar a reutilização por parte do Contexto que permite escolher entre uma família de algoritmos que possuem funcionalidades em comum; os algoritmos em classes Strategy possuem variações do seus algoritmos independentemente do seu contexto, assim é mais fácil utilizá-los, trocá-los, compreende-los e estende-los; diminuição ou eliminação da lógica condicional clarificando ainda mais os algoritmos; a Strategy permite que se escolham diferentes implementações do mesmo comportamento; utilizando Strategy há uma grande simplificação na classe ao mover variações de um algoritmo para uma hierarquia; habilita-se que um algoritmo seja substituído por outro em tempo de execução.
- As desvantagens na utilização do Padrão Strategy é a complicação que há de como os algoritmos obtêm ou recebem dados de suas

classes de contexto; o cliente deve conhecer como que os Strategies diferem, antes mesmo que ele possa selecionar um mais apropriado para o contexto da aplicação; o custo da comunicação entre o contexto e o Strategy é significativo, dado que os Strategies concretos não necessariamente usarão todas as informações da Strategy abstrata, portanto podem haver situações em que o contexto criará e inicializará parâmetros que nunca serão usados; Strategies aumentam o número de objetos no sistema, que pode ser ruim em determinadas situações em termos de custo e por fim pessoas inexperiente podem ter dificuldade sobre o funcionamento do código por não entender o que é e como funciona o padrão.

Implementação

A implementação a ser demonstrada é referente a área de telecomunicações onde uma grande empresa da área de telecomunicações possui uma intranet de atendimento para todo o Brasil e teve num dos seus módulos internos um problema que foi resolvido com o padrão de projeto Strategy. A intranet de atendimento dessa grande empresa de telecomunicações possui todo o gerenciamento interno da empresa, como cadastros e pesquisas dos clientes e funcionários, planos de celulares disponibilizados, ofertas e promoções do dia, mês e ano, aparelhos celulares em vigência pela operadora, áreas de comunicação entre os atendentes e seus supervisores, entre diversos outros. Essa intranet de atendimento é utilizada principalmente pelos funcionários que prestam atendimentos em CallCenters, Ilhas e Segmentos. O CallCenter é um centro de atendimento ao consumidor externo, as Ilhas são atendimentos internos aos operadores de CallCenters que precisam tirar alguma dúvida e os Segmentos são operadores mais internos que cuidam de dúvidas mais específicas como configuração de celulares, ou a área financeira, etc. Cada CallCenter atende a determinados Estados e Municipios do país, cada Ilha atende apenas um certo número de atendentes de CallCenters e cada segmento atende a um determinado número de Ilhas.

O módulo específico que será tratado no exemplo abaixo é o módulo de Atendentes do sistema. Este módulo é responsável por configurar num contexto do sistema as informações do atendente que estava acessando o sistema. Dependendo do atendente que se autentica e o seu nível de atendimento (CallCenter, Ilha, Segmento) o sistema seta algumas configurações importantes que definem o layout que será apresentado e quais os acessos que esse atendente possui no sistema.

Neste módulo havia a seguinte lógica condicional.

Listagem 1: Lógica condicional da aplicação.

```
if(obj.equals(“AtendenteCallCenter”)){
    params = new Object[4];
    params[0] = getCallCenter();
    params[1] = getLogin();
    params[2] = getInicioVigencia();
    params[3] = getOrigemCadastro();
}
if(obj.equals(“AtendenteIlha”)){
    params = new Object[5];
    params[0] = getCallCenter();
    params[1] = getLogin();
    params[2] = getInicioVigencia();
    params[3] = getOrigemCadastro();
    params[4] = getIlha();
}
if(obj.equals(“AtendenteSegmento”)){
    params = new Object[6];
    params[0] = getCallCenter();
    params[1] = getLogin();
    params[2] = getInicioVigencia();
    params[3] = getOrigemCadastro();
    params[4] = getIlha();
    params[5] = getSegmento();
}
```

Para cada tipo de atendente o sistema configura os parâmetros necessários para um atendente específico. Para cada nova configuração ou caso surja um novo atendente ou uma nova lógica para um atendente, tudo isso deve ficar dentro dessa condicional, podendo assim, aumentar a lógica condicional e tornando esta cada vez mais complexa e difícil de ser mantida. Esse código contém muitos estados, portanto indica à necessidade de aplicar o padrão Strategy. Uma observação importante é que algumas verificações extensas foram retiradas de partes internas de cada uma das verificações acima para não tornar o código mais complexo e maior.