Padrões GoF

Grupo 1

João Paulo Nunes Pereira (201610658)

Luan Martins Correa (202023809)

Nicolas Prates Anacleto (201816957)

Salomão Quevedo Gerard da Luz (201719382)

Thailon dorneles (202014087)

Vinicius Faitão (202065022)

Estruturais

Adapter

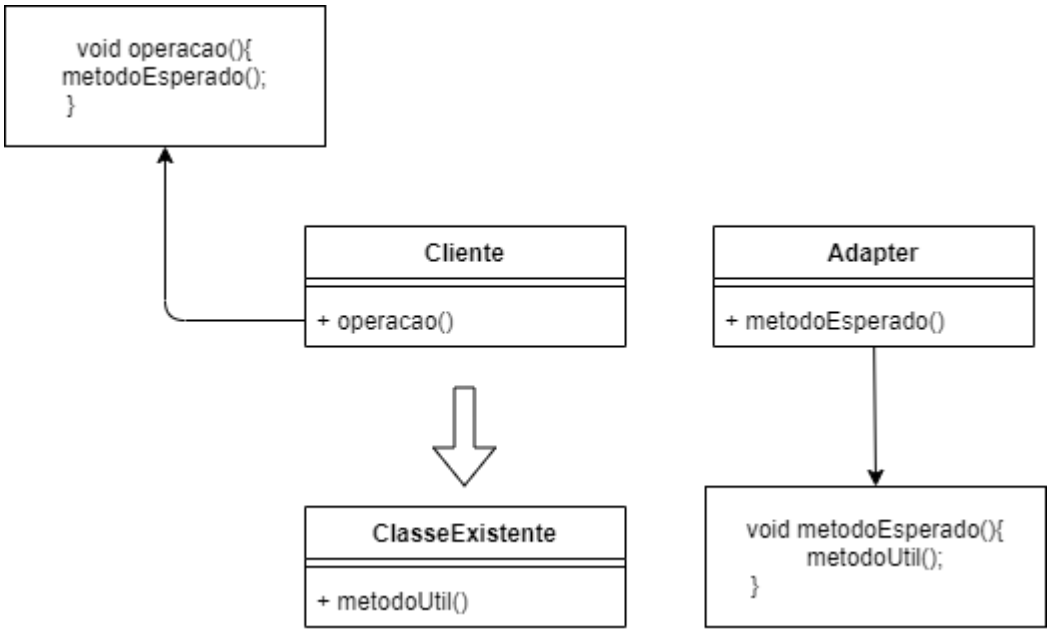
Descrição do Padrão:

Permite a comunicação entre classes que não poderiam trabalhar juntas devido a incompatibilidade de suas interfaces.

Exemplo de uso do Padrão:

A classe Adaptador deve ser usada quando você quer usar uma classe existente, mas sua interface não for compatível com o resto do seu código.
O padrão Adapter permite que você crie uma classe de meio termo que serve como um tradutor entre seu código e a classe antiga, uma classe de terceiros, ou qualquer outra classe com uma interface estranha.

Modelo de implementação:



Correlações com outros padrões:

O adapter tem uma ideia parecida com o proxy, os dois fazem um meio entre as duas classes, porém o adapter fornece uma interface diferente para um objeto encapsulado, o Proxy fornece a ele a mesma interface

Bridge

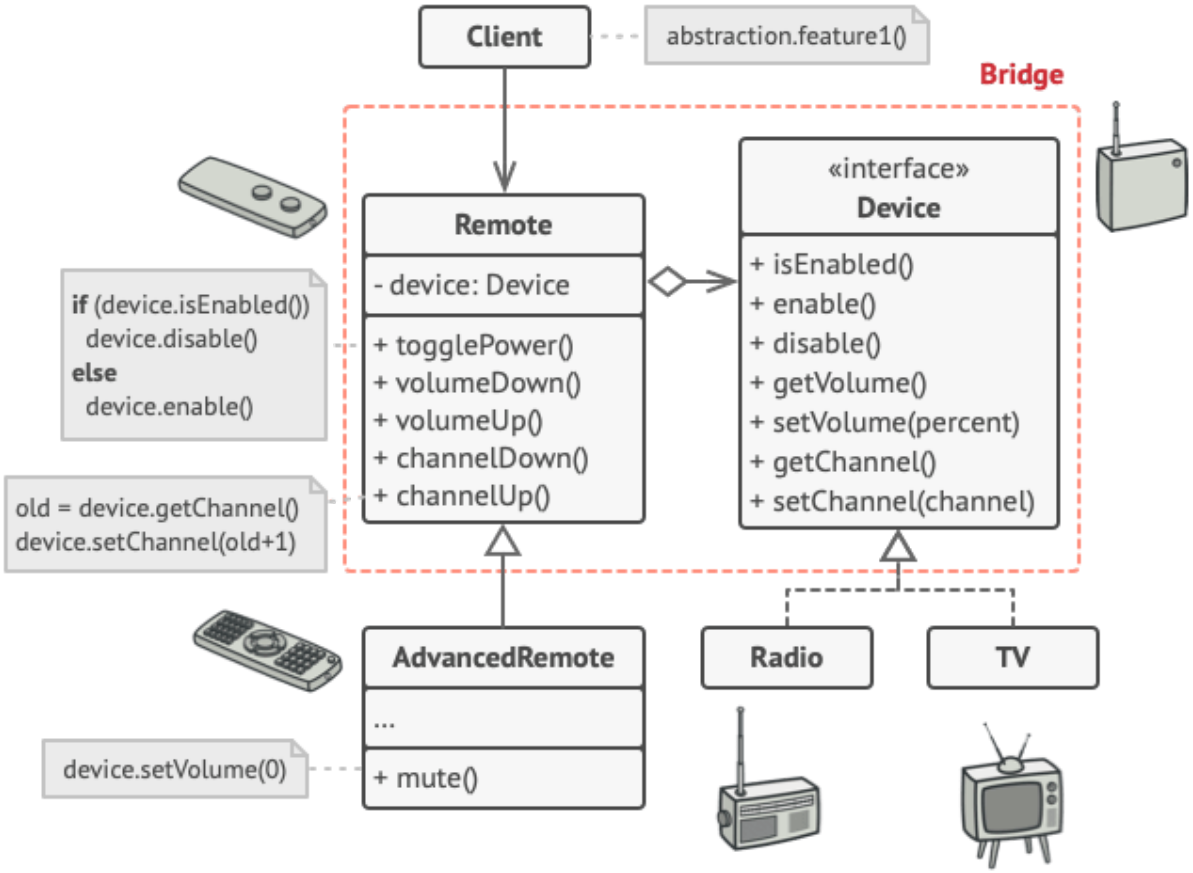
Descrição do Padrão:

Este é um padrão utilizado quando é desejável que uma interface possa variar independentemente das suas implementações, ou seja, permite que você divida uma classe grande ou um conjunto de classes intimamente ligadas em duas hierarquias separadas.

Exemplo de uso do Padrão:

Podemos usar no nosso projeto semestral. Pois ao ter uma classe Pessoa, ao invés de toda pessoa “Administrador, Voluntário, PetOwner” ser uma classe diferente e para cada classe diferente termos um Contact, fazemos uma implementação bridge, onde uma classe Pessoa receberá um tipo definindo se é “Administrador, Voluntário, PetOwner” e será ligado com um único Contact.

Modelo de implementação:



Correlações com outros padrões:

Não

Composite

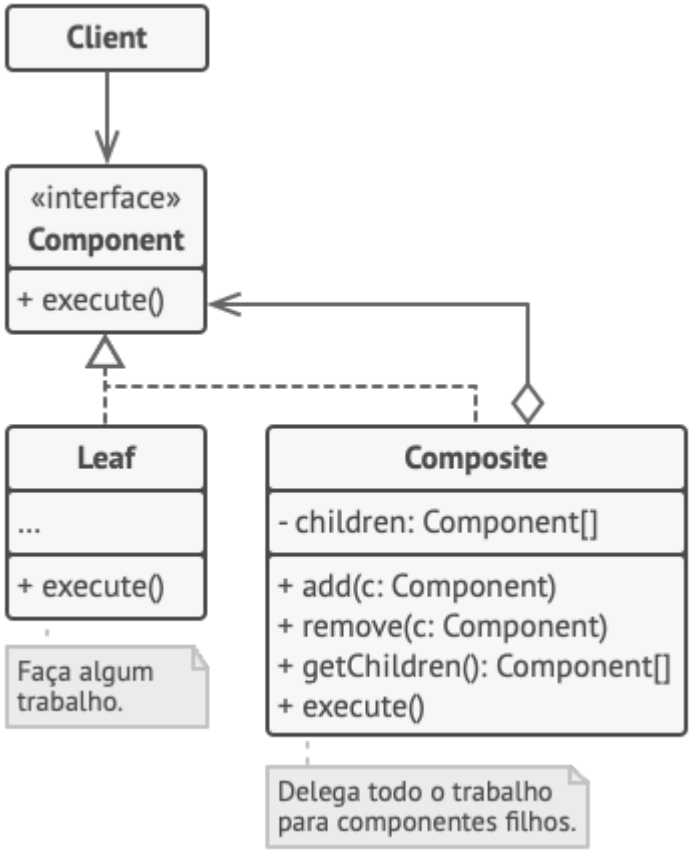
Descrição do Padrão:

Este é um padrão estrutural que permite que você componha objetos em estruturas de árvores e então trabalhe com essas estruturas como se fossem objetos individuais.

Exemplo de uso do Padrão:

Neste exemplo podemos pensar em uma ação por etapas, onde todas as classes são implementadas com a mesma interface.Quando for chamado um método, os próprios objetos passam o valor pela árvore. Basicamente no sistema de pet, não precisamos saber se o objeto pessoa é um administrador ou um assistente para saber seu endereço, você apenas trata todos os objetos com a mesma interface quando quisermos saber o endereço do usuário.

Modelo de implementação:



Correlações com outros padrões:

Não

Decorator

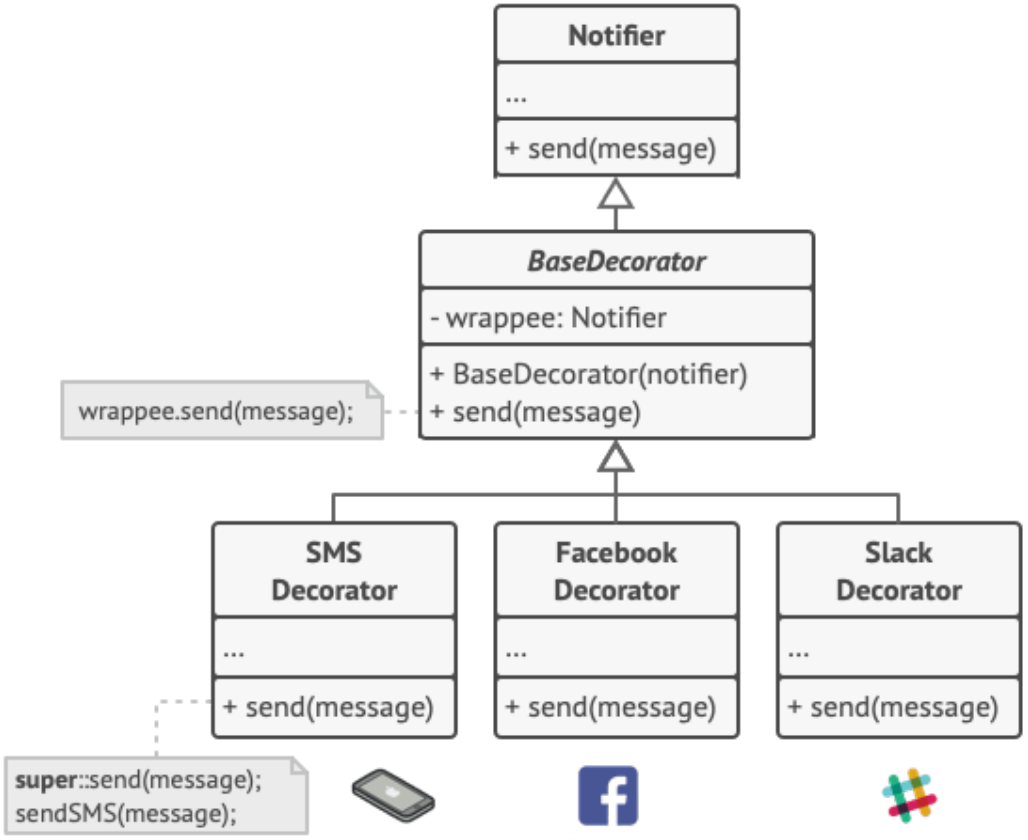
Descrição do Padrão:

Este é um padrão estrutural que permite que você acople novos comportamentos para objetos ao colocá-los dentro de invólucros de objetos que contém os comportamentos.

Exemplo de uso do Padrão:

Neste exemplo podemos pensar em 1 único objeto base que tem a função de envio de notificações, nele é recebido qual o tipo de ambiente utilizado para fazer a notificação e qual a mensagem. Assim, quando o usuário escolher receber uma notificação por Facebook ou Email, ou até mesmo ambos, será feito um decorator para cada tipo de notificador, 1 para facebook e 1 para email que todos são utilizados pela classe base de notificação.

Modelo de implementação:



Correlações com outros padrões:

Não

Facade

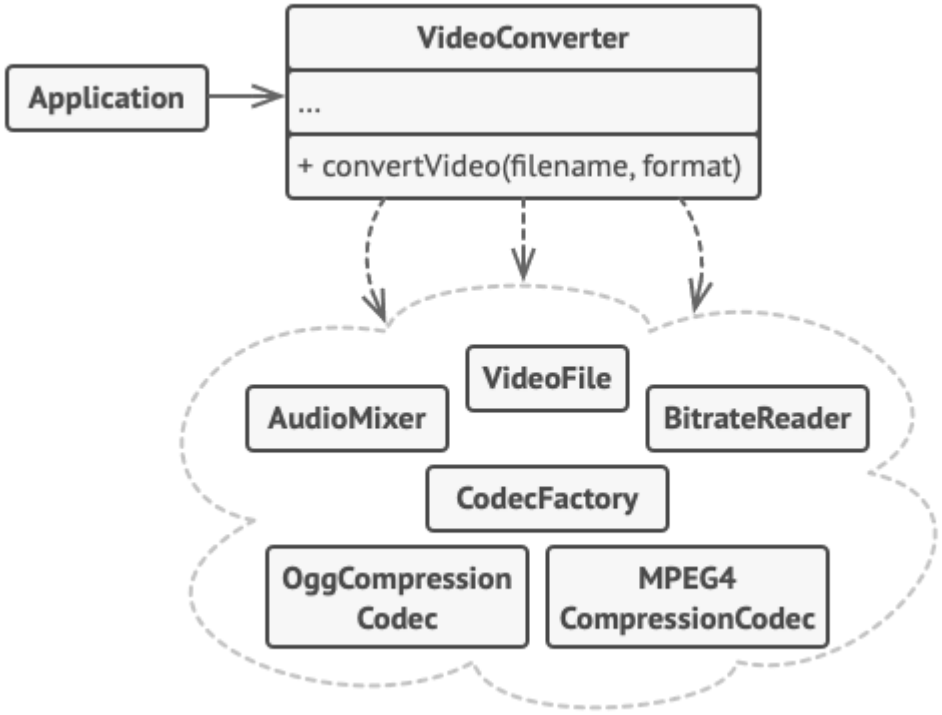
Descrição do Padrão:

Este é um padrão estrutural que fornece uma interface simplificada para uma biblioteca, um framework ou qualquer conjunto complexo de classes.

Exemplo de uso do Padrão:

Como exemplo temos o uso de uma classe de conversão de vídeo, essa classe VideoConverter é o facade, onde nele se é chamado todas as outras classes que será utilizada, para que ao invés de ter um código gigante, fazemos separações e utilizações dessas classes de forma componentizada, onde cada classe tem sua função armazenada e utilizada pelo facade.

Modelo de implementação:



Correlações com outros padrões:

Não

Flyweight

Descrição do Padrão:

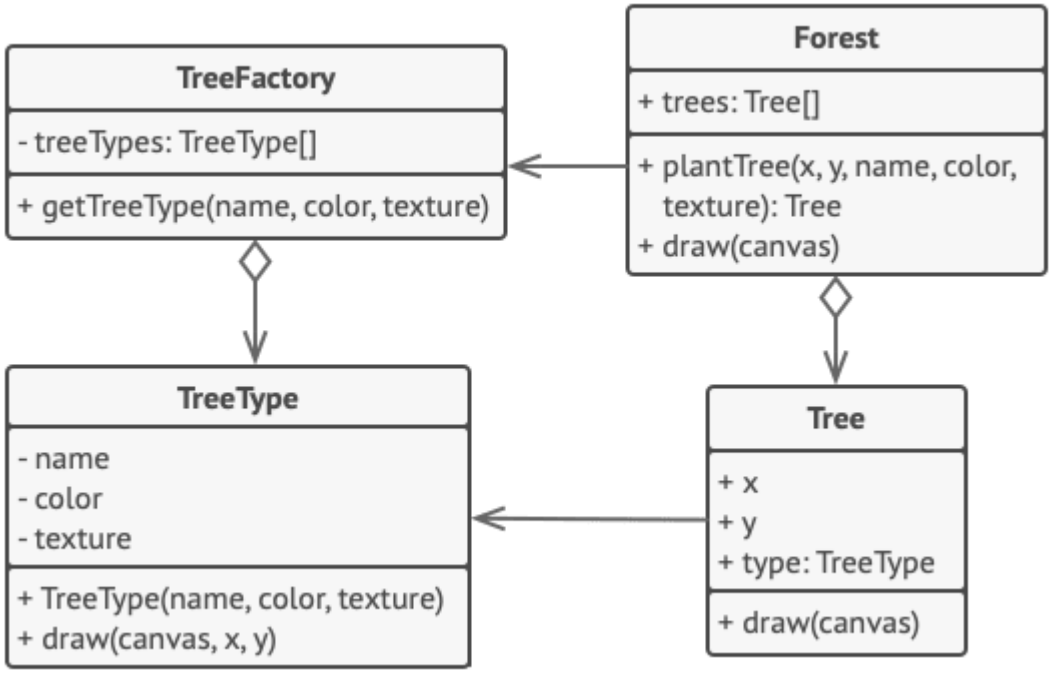
Flyweight é um padrão de projeto de software que serve para quando se quer manipular vários objetos em memória, sendo que vários possuem informações repetidas. Dado que a memória é limitada, é possível segregar a informação repetida em um objeto adicional que atenda às características de imutabilidade e comparabilidade que seja possível comparar com outro objeto para determinar se ambos carregam a mesma informação.

Exemplo de uso do Padrão:

Um exemplo é o processador de texto. Cada caractere representa um objeto que possui uma família de fonte, um tamanho de fonte e outras informações sobre o símbolo. Como imaginado, um documento grande com tal estrutura de dados facilmente ocuparia toda a memória disponível no sistema. Para resolver o problema, como muitas dessas informações são repetidas, o flyweight é usado para reduzir os dados. Cada objeto de caractere contém uma referência para outro objeto com suas respectivas propriedades.

Modelo de implementação:

No modelo abaixo, o padrão Flyweight ajuda a reduzir o uso de memória quando renderizando milhões de objetos árvore em uma tela.



O padrão extrai o estado intrínseco repetido para um classe Arvore principal e o move para dentro da classe flyweight Tipo Arvore.

Correlações com outros padrões:

Não

Proxy

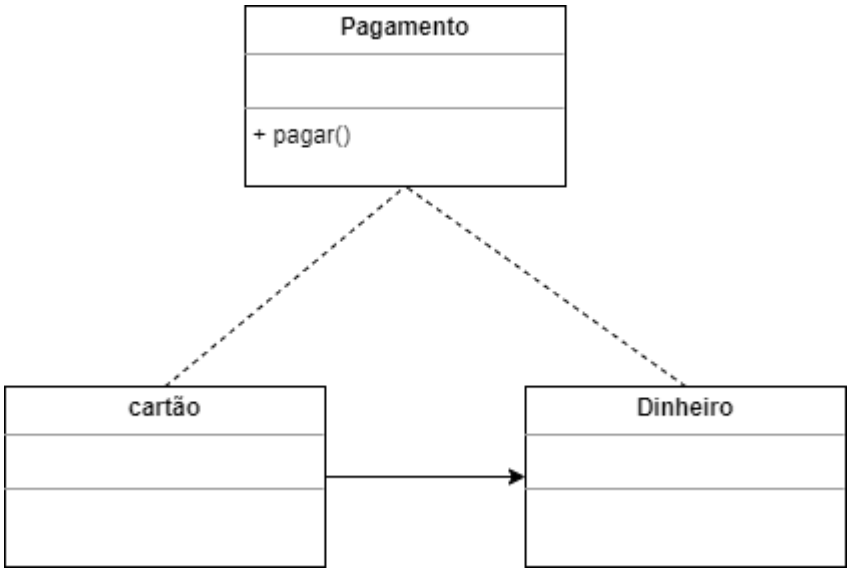
Descrição do Padrão:

O proxy serve como um espelho da classe com as mesmas funções do serviço original, assim que chamado cria uma "cópia" e delega todo o serviço para ele.
Pode se dizer que ele é um filtro também pois recebe os dados e depois passa para o serviço.

Exemplo de uso do Padrão:

Um cartão de crédito é um proxy para uma conta bancária, que é um proxy para uma porção de dinheiro. Ambos implementam a mesma interface porque não há necessidade de carregar uma porção de dinheiro por aí.

Modelo de implementação:



Correlações com outros padrões:

O Proxy tem uma ideia parecida com o adapter, os dois fazem um meio entre as duas classes, porém o adapter fornece uma interface diferente para um objeto encapsulado, o Proxy fornece a ele a mesma interface .

Padrões GoF

Comportamentais

Chain of Responsibility

Descrição do Padrão:

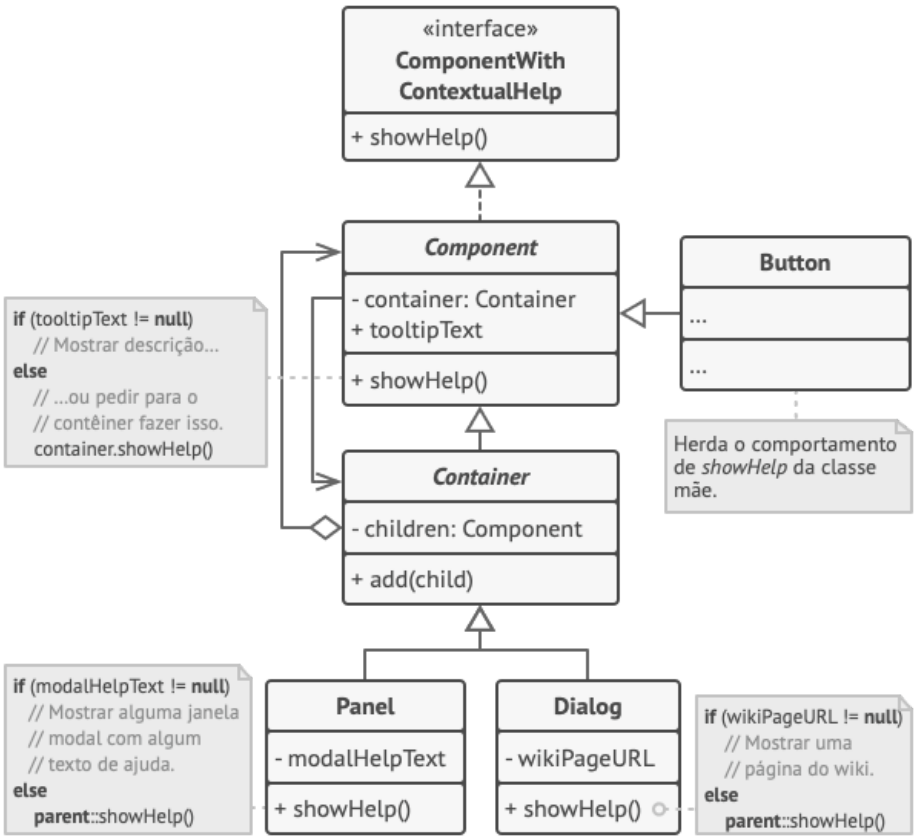
O Chain of Responsibility é um padrão de projeto comportamental que permite que você passe pedidos por uma corrente de handlers. Ao receber um pedido, cada handler determina se processa o pedido ou o passa adiante para o próximo handler na corrente.

Exemplo de uso do Padrão:

Um exemplo da aplicação desse padrão é o mecanismo de herança nas linguagens orientadas a objeto: um método chamado em um objeto é buscado na classe que implementa o objeto e, caso não seja encontrado na superclasse desta classe de maneira recursiva. Como benefício do uso do padrão, ele permite determinar quem será o objeto que irá tratar a requisição durante a execução. Cada objeto pode tratar ou passar a mensagem para o próximo na cadeia. Dessa forma, o acoplamento é reduzido, dando ainda flexibilidade adicional na atribuição de responsabilidades a objetos.

Modelo de implementação:

No modelo abaixo, o padrão Chain of Responsibility é responsável por mostrar informação de ajuda contextual para elementos de GUI ativos.



O GUI da aplicação é geralmente estruturado como uma árvore de objetos. Por exemplo, a classe Dialog, que renderiza a janela principal da aplicação, seria a raíz do objeto árvore. O dialog contém Painéis, que podem conter outros painéis ou simplesmente elementos de baixo nível como button e camposdetexto.

Correlações com outros padrões:

Não

Command

Descrição do Padrão:

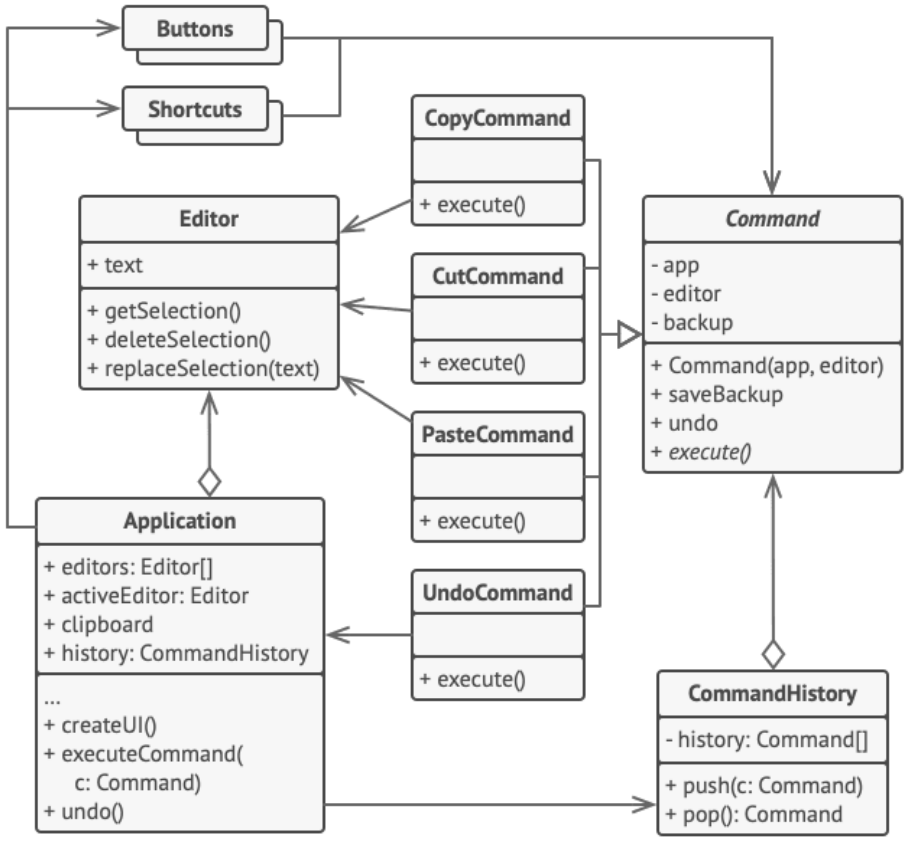
O Command é um padrão de projeto comportamental que transforma um pedido em um objeto independente que contém toda a informação sobre o pedido. Essa transformação permite que você parametrize métodos com diferentes pedidos, atrase ou coloque a execução do pedido em uma fila, e suporte operações que não podem ser feitas.

Exemplo de uso do Padrão:

O padrão Command é muito comum em códigos da linguagem Java. Em grande parte, é usado como uma alternativa para retornos de chamada para ser parâmetro de elementos da interface do usuário com ações. Também é usado para tarefas de enfileiramento, rastreamento de histórico de operações etc.
Alguns exemplos do padrão Command nas principais bibliotecas Java:
Todas as implementações de java.lang.Runnable
Todas as implementações de javax.swing.Action

Modelo de implementação:

O padrão Command ajuda a manter um registro da história de operações executadas e torna possível reverter uma operação se necessário.



Correlações com outros padrões:

Não

Interpreter

Descrição do Padrão:

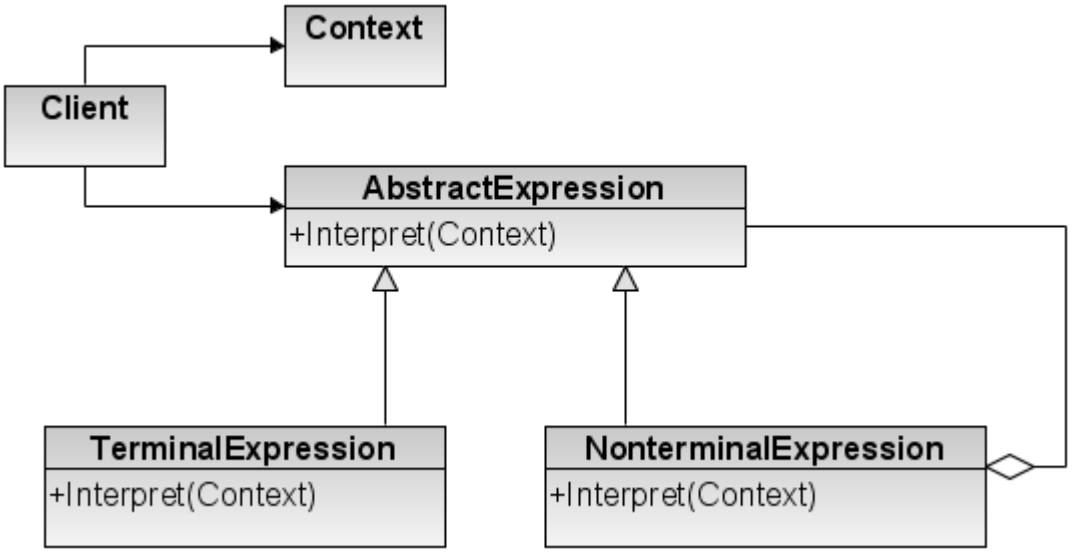
Interpreter é um dos padrões de projeto de software, famosos como "Design Patterns", muito utilizado para a resolução de problemas quando a modelagem de sistemas ou softwares. Esse padrão está incluso na categoria de Padrão Comportamental, ou seja, ele busca solucionar problemas de modelagem que tratam o comportamento de classes.

Exemplo de uso do Padrão:

O padrão Interpreter define uma representação gramatical de uma linguagem e um intérprete para realizar a interpretação da gramática. Os músicos são exemplos de intérpretes. O tom de um som e sua duração podem ser representados em notação musical em uma pauta. Esta notação fornece a linguagem da música. Músicos tocando a música da partitura são capazes de reproduzir o tom original e duração de cada som representado.

Modelo de implementação:

O padrão Interpreter sugere modelar o domínio com uma gramática recursiva. Cada regra na gramática é tanto um 'composite' (uma regra que referencia outras regras) ou um 'terminal' (uma folha/nó numa estrutura de árvore). O Interpreter baseia-se na travessia recursiva do padrão Composite para interpretar as 'sentenças' que ele deve processar.



Correlações com outros padrões:

Não

Iterator

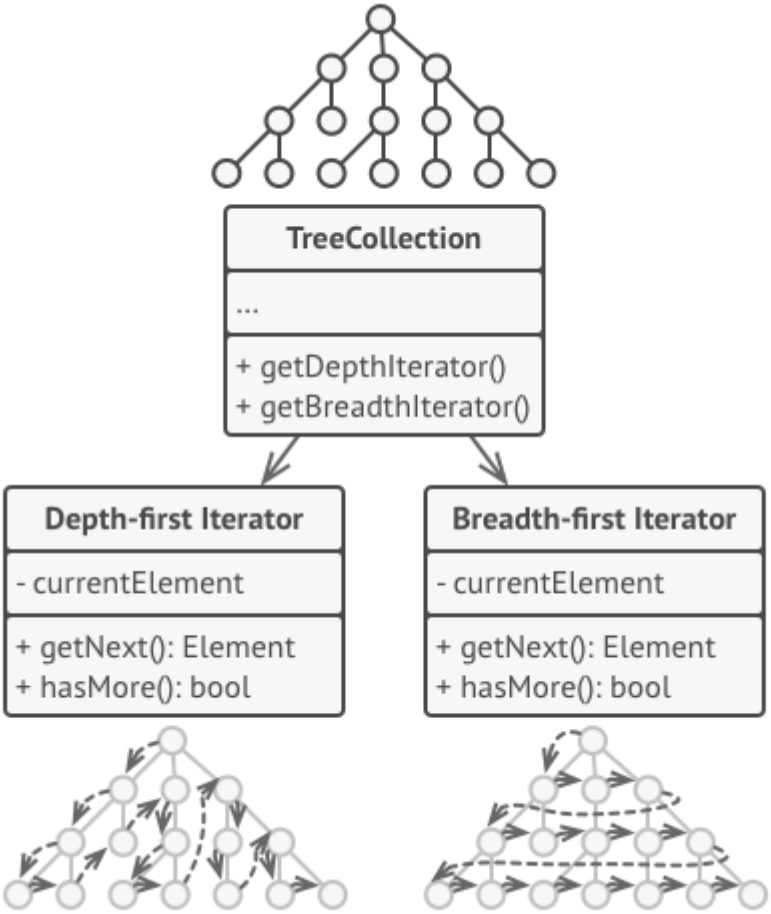
Descrição do Padrão:

O Iterator é um padrão de projeto comportamental que permite percorrer de uma forma predefinida ou aleatória os elementos de pilhas, árvores, listas, etc.

Exemplo de uso do Padrão:

O objeto iterador guarda todos os detalhes do trajeto, como a posição atual e quantos elementos faltam para chegar ao fim. Podemos dizer que ele é uma espécie de organizador.

Modelo de implementação:



Correlações com outros padrões:

Não

Memento

Descrição do Padrão:

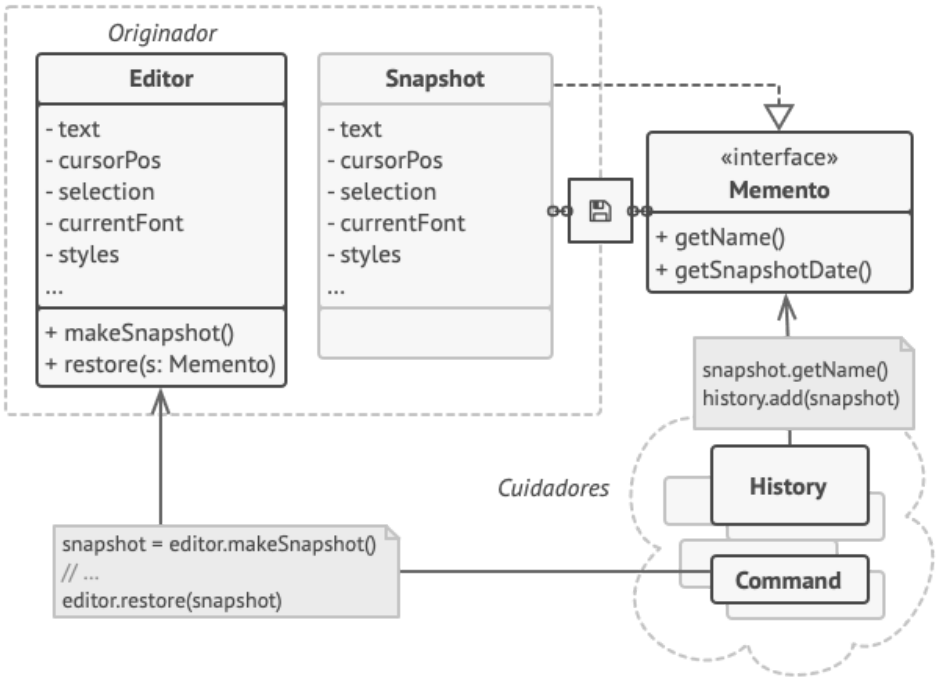
O Memento permite o salvamento e restauração do estado anterior de um objeto sem revelar os detalhes de sua implementação.

Exemplo de uso do Padrão:

O Memento é usado para “copiar” o estado de um objeto e armazená-lo separadamente, tornando possível ser feita a restauração do estado anterior do mesmo. O caso mais clássico e conhecido do uso deste padrão, percebe-se no uso do “desfazer”, que retorna o estado anterior de determinado contexto, revertendo uma operação.

Modelo de implementação:

Define-se a classe que fará o papel de originadora, que também pode ser múltiplos objetos ao invés de um único objeto central. Com isso, a classe ‘Memento’ é criada, juntamente aos seus campos que espelham os campos da classe originadora. A classe ‘Memento’ também é imutável. Na classe originadora é adicionado o método para a produção de ‘mementos’, e também outro para realizar a restauração. A classe originadora deve passar seu estado para ‘Memento’.



Correlações com outros padrões:

Não

Observer

Descrição do Padrão:

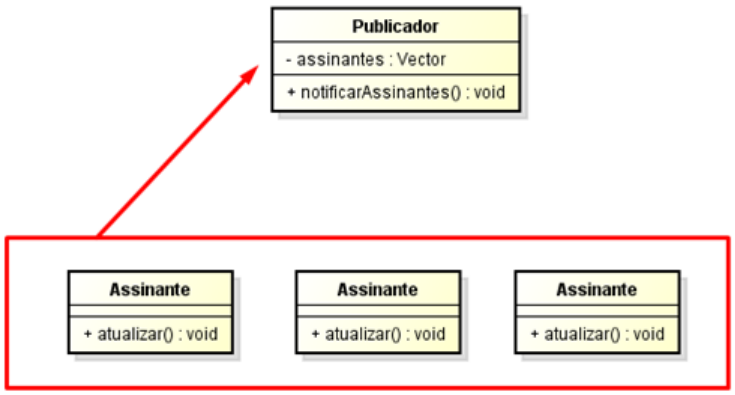
O Observer é um padrão de projeto comportamental que permite que você defina um mecanismo de assinatura para notificar múltiplos objetos sobre quaisquer eventos que aconteçam com o objeto que eles estão observando.

Exemplo de uso do Padrão:

Se você assinar um jornal ou uma revista, você não vai mais precisar ir até a banca e ver se a próxima edição está disponível. Será papel da publicadora enviar novas edições para você, já que é um assinante.

A publicadora mantém uma lista de assinantes e sabe em quais revistas eles estão interessados. Os assinantes podem deixar essa lista a qualquer momento quando desejarem que a publicadora pare de enviar novas revistas para eles.

Modelo de implementação:



Correlações com outros padrões:

O Observer tem várias maneiras de se relacionar com outros padrões, como o Chain of Responsibility, Command e Mediator.

O Chain of Responsibility passa um pedido sequencialmente ao longo de uma corrente dinâmica de potenciais destinatários até que um deles atue no pedido.

O Command estabelece conexões unidirecionais entre remetentes e destinatários.

O Mediator elimina as conexões diretas entre remetentes e destinatários, forçando-os a se comunicar indiretamente através de um objeto mediador.

Mediator

Descrição do Padrão:

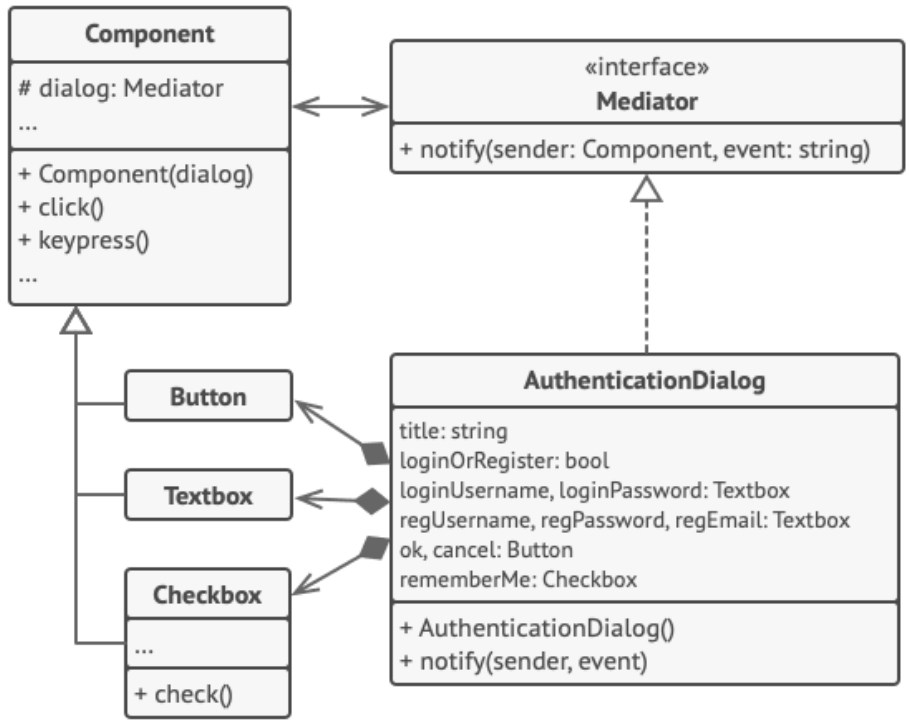
O Mediator permite que você reduza as dependências caóticas entre objetos. O padrão restringe comunicações diretas entre objetos e os força a colaborar apenas através do objeto mediador.

Exemplo de uso do Padrão:

O Mediator é utilizado quando existe dificuldade para alterar alguma classe, devido ao seu possível acoplamento a várias outras classes. Ao aplicar este padrão, é possível obter todas as relações entre classes para uma classe “a parte”, separando as possíveis alterações para um componente específico em relação aos outros. A partir disso a comunicação é sempre feita de forma mediada entre os componentes (pela classe mediadora).

Modelo de implementação:

A implementação é feita com base em um grupo de classes fortemente acopladas, de forma a deixa-las mais independentes. Isso é feito com a declaração da interface do ‘mediador’ para que possa ser feito o reuso de classes componentes em variados contextos. Os componentes devem possuir uma referencia ao objeto mediador, que geralmente é definida no construtor.
Neste exemplo, o padrão ‘Mediator’ ajuda a eliminar dependências repetidas de várias classes UI: botões, caixas de seleção, e textos de rótulos:



Correlações com outros padrões:

Assemelha-se ao Observer, podendo na maioria das vezes implementa qualquer um destes (Mediator/Observer), ou até ambos. O Mediator foca em tentar eliminar múltiplas

dependências, já o Observer foca em estabelecer comunicações dinâmicas com base em subordinação.

State

Descrição do Padrão:

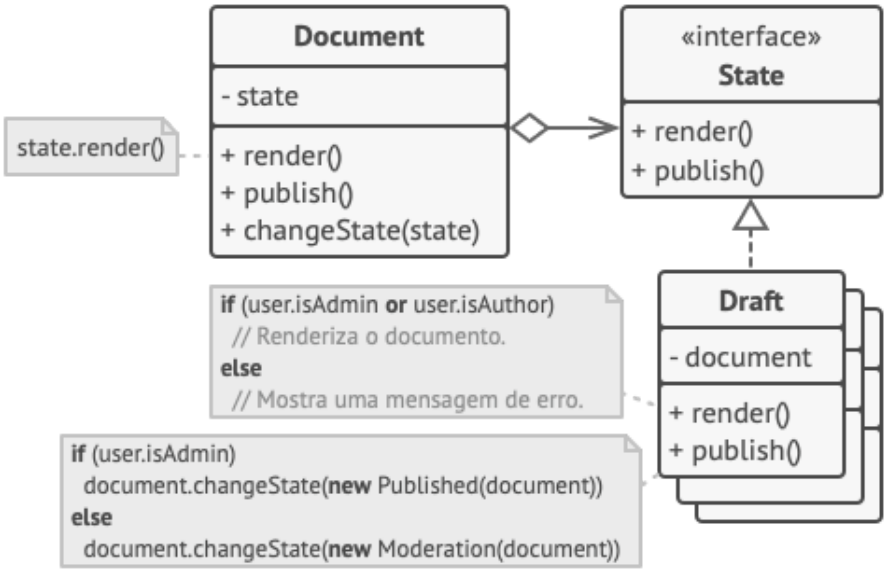
O State permite que um objeto altere seu comportamento quando seu estado interno muda, como se o objeto mudasse de classe.

Exemplo de uso do Padrão:

Usa-se esse padrão quando um determinado objeto se comporta de forma diferente de acordo com seu estado, especialmente quando são muitos estados ou quando mudam de forma frequente.

Modelo de implementação:

O State supõe que sejam criadas classes para cada estados possível de um objeto, contendo todos os comportamentos de estados nestas classes. No objeto original é armazenado uma referencia para um dos estados (objeto) correspondentes ao seu determinado contexto.



Correlações com outros padrões:

Apresenta semelhança com o padrão Strategy, onde ambos apresentam o comportamento de definir papeis para os objetos auxiliares. No ‘Strategy’ esses objetos são independentes, já no ‘State’ não existe restrições de dependências estados, permitindo alterações nos mesmos.

Strategy

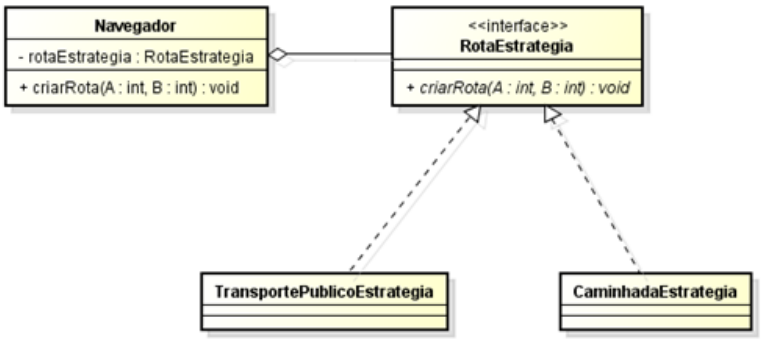
Descrição do Padrão:

O **Strategy** é um padrão de projeto comportamental que permite que você defina uma família de algoritmos, coloque-os em classes separadas, e faça os objetos deles intercambiáveis.

Exemplo de uso do Padrão:

Imagine uma aplicação de navegação onde a ideia inicial era voltada para viajantes casuais e era preciso ter a funcionalidade planejamento automático de rotas. Ao longo do tempo, foram adicionadas a funcionalidade de calcular rotas sobre rodovias, depois rotas de caminhadas, rotas de transporte público e mais tarde rotas para ciclistas. O Strategy nesse caso seria um ótimo padrão, pois sugere que você pegue uma classe que faz algo específico em diversas maneiras diferentes e extraia todos esses algoritmos para classes separadas chamadas *estratégias*. A classe original, chamada *contexto*, deve ter um campo para armazenar uma referência para um dessas estratégias. O contexto delega o trabalho para um objeto estratégia ao invés de executá-lo por conta própria.

Modelo de implementação:



Correlações com outros padrões:

A estrutura do Strategy é muito parecido com a de outros padrões como Brigde e State, pois todos são baseados em delegar o trabalho para outros objetos, porém resolvem problemas diferentes. O Decorator permite que você mude a pele de um objeto, enquanto o Strategy permite que você mude suas entranhas.O State pode ser considerado como uma extensão do Strategy. Ambos padrões são baseados em composição.

Visitor

Descrição do Padrão:

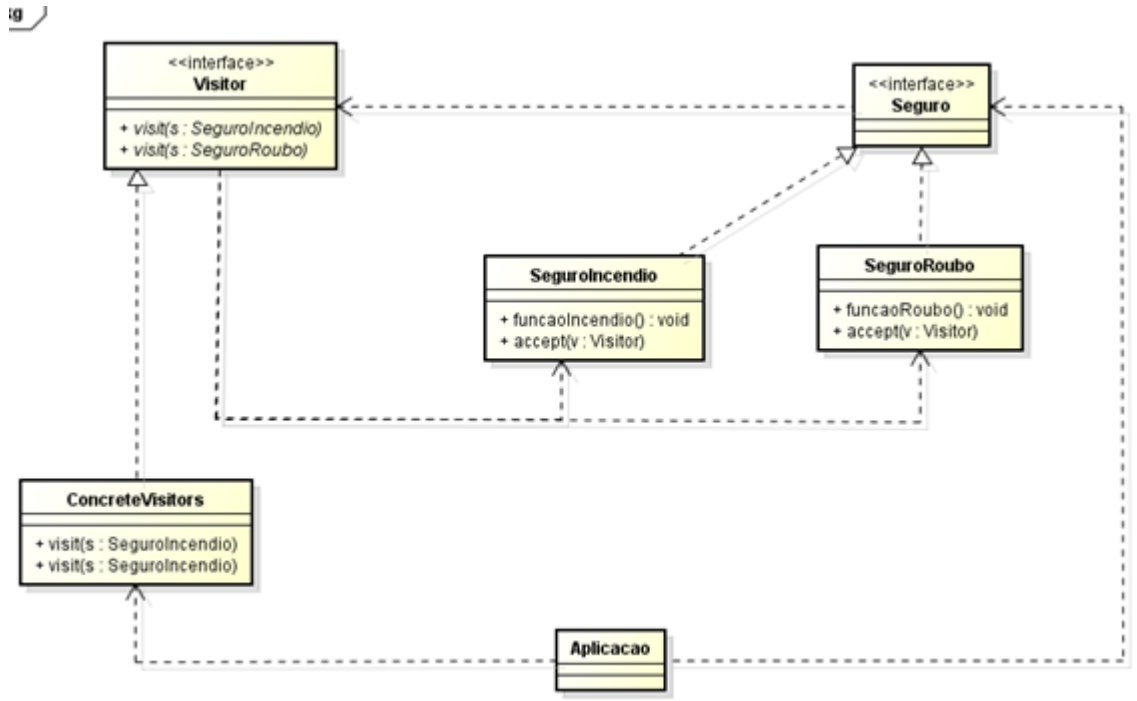
O **Visitor** é um padrão de projeto comportamental que permite que você separe algoritmos dos objetos nos quais eles operam.

O padrão Visitor sugere que você coloque o novo comportamento de uma aplicação em uma classe separada chamada *visitante*, ao invés de tentar integrá-lo em classes já existentes. O objeto original que teve que fazer o comportamento é passado para um dos métodos da visitante como um argumento, desde que o método acesse todos os dados necessários contidos dentro do objeto.

Exemplo de uso do Padrão:

Imagine um agente de seguros experiente que está ansioso para obter novos clientes. Ele pode visitar cada prédio de uma vizinhança, tentando vender apólices de seguro para todos que encontra e dependendo do tipo de organização que o ocupa o prédio ele vai oferecer apólices especializadas.

Modelo de implementação:



Correlações com outros padrões:

O Visitor pode ser considerado uma versão mais poderosa do Command. Também podemos usar o Visitor para executar operações em uma árvore Composite. Podemos usar junto com o Iterator para percorrer uma estrutura de dados e executar operações sobre seus elementos.